

Inhaltsverzeichnis

Tag 1 Eine Handvoll Java

[Was ist Java?](#)

[Warum lesen Sie dieses Buch?](#)

[Eintauchen in die Java-Programmierung](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 2 Objektorientierte Programmierung - ein erster Eindruck

[In Objekten denken](#)

[Objekte und Klassen](#)

[Klassen und deren Verhalten organisieren](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 3 Das Java-ABC

[Anweisungen und Ausdrücke](#)

[Variablen und Datentypen](#)

[Kommentare](#)

[Literale](#)

[Ausdrücke und Operatoren](#)

[String-Arithmetik](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 4 Arbeiten mit Objekten

[Erstellen neuer Objekte](#)

[Speichermanagement](#)

[Verwenden von Klassen- und Instanzvariablen](#)

[Aufrufen von Methoden](#)

[Klassenmethoden](#)

[Referenzen auf Objekte](#)

[Casting und Konvertieren von Objekten und Primitivtypen](#)

[Objekte vergleichen und mehr](#)

[Bestimmen der Klasse eines Objekts](#)

[Klassen und Methoden mit Reflexion inspizieren](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 5 Arrays, Bedingungen und Schleifen

[Arrays](#)

[switch-Bedingungen](#)

[for-Schleifen](#)

[while- und do-Schleifen](#)

[Unterbrechen von Schleifen](#)

[Benannte Schleifen](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 6 Java: Eine Klassensprache

[Definieren von Klassen](#)

[Erstellen von Instanz- und Klassenvariablen](#)

[Erstellen von Methoden](#)

[Entwickeln von Java-Applikationen](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 7 Mehr über Methoden

[Finalizer-Methoden](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 8 Grundlagen der Java-Applets

[Unterschiede zwischen Applets und Anwendungen](#)

[Sicherheitseinschränkungen von Applets](#)

[Eine Java-Version wählen](#)

[Erstellen von Applets](#)

[Zusammenfassung](#)

Tag 9 Programme mit Grafik, Fonts und Farbe verfeinern

[Die Klasse Graphics](#)

[Zeichnen und Füllen](#)

[Text und Schriften](#)

[Farbe](#)

[Fortgeschrittene Grafikoperationen mit Java2D](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 10 Bilder, Sound und Animation

[Animationen unter Java erstellen](#)

[Das Flimmern in Animationen reduzieren](#)

[Bilder laden und anzeigen](#)

[Animationen mit Bildern](#)

[Klänge laden und verwenden](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 11 Einfache Benutzeroberflächen für Applets

[Das Abstract Windowing Toolkit](#)

[Die Basiskomponenten der Benutzeroberfläche](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 12 Benutzerschnittstellen entwerfen

[Das elementare Layout einer Benutzerschnittstelle](#)

[Verschiedene Layout-Manager gleichzeitig](#)

[Fortgeschrittene Layout-Manager](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 13 Ereignisverarbeitung in Applets

[Ereignisverarbeitung](#)

[Mausklicks behandeln](#)

[Mausbewegungen behandeln](#)

[Behandlung von Tastaturereignissen](#)

[Beispiel: Zeichen eingeben, anzeigen und versetzen](#)

[Der generische Eventhandler](#)

[Handhabung von Ereignissen der Benutzeroberfläche](#)

[Beispiel: Hintergrundfarbwechsler](#)

[Ereigniscode einfügen](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 14 Fortgeschrittene Benutzeroberflächen mit dem AWT

[Fenster, Frames und Dialogfelder](#)

[Menüs](#)

[AWT-Stand-alone-Applikationen erstellen](#)

[Komplettes Beispiel: RGB/HSB-Konverter](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 15 Pakete, Schnittstellen und mehr

[Modifier](#)

[Zugriffskontrolle für Methoden und Variablen](#)

[Vier Schutzebenen](#)

[Konventionen für den Zugriff auf Instanzvariablen](#)

[Zugriffskontrolle und Vererbung](#)

[Accessor-Methoden](#)

[Klassenvariablen und -methoden](#)

[Der final-Modifier](#)

[abstract-Methoden und -Klassen](#)

[Verzeichnisstruktur definieren](#)

[Pakete und Klassenschutz](#)

[Was sind Schnittstellen?](#)

[Das Problem der Einfachvererbung](#)

[Schnittstellen und Klassen](#)

[Interne Klassen](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 16 Ausnahmezustände: Fehlerbehandlung und Sicherheit

[Programmieren im großen](#)

[Programmieren im kleinen](#)

[Einschränkungen beim Programmieren](#)

[Die finally-Klausel](#)

[Digitale Signaturen zur Identifikation von Applets](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 17 Java und Streams

[Eingabedatenstreams und Reader](#)

[Ausgabedatenstreams und Writer](#)

[Zusammenhängende Klassen](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 18 Kommunikation über das Internet

[Netzwerkprogrammierung in Java](#)

[Trivia: Ein einfacher Socket-Client und -Server](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 19 JavaBeans und andere fortgeschrittene Features

[Applet-Tricks](#)

[Ausschneiden, Kopieren und Einfügen](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 20 Benutzerschnittstellen mit Swing

[Die Vorteile von Swing](#)

[Der Umgang mit Swing](#)

[Neue Features von Swing](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Tag 21 Ereignisbehandlung mit Swing

[Das Hauptereignis](#)

[Mit Methoden arbeiten](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

Anhang A Java-Sprachübersicht

[Reservierte Wörter](#)

[Kommentare](#)

[Literale](#)

[Variablendeklaration](#)

[Wertzuweisung an Variablen](#)

[Operatoren](#)

[Objekte](#)

[Arrays](#)

[Schleifen und Bedingungen](#)

[Klassendefinitionen](#)

[Methoden- und Konstruktor-Definitionen](#)

[Import](#)

[Überwachung](#)

Anhang B Die Java-Klassenbibliothek

[java.applet](#)

[java.awt](#)

[Java.awt.color](#)

[java.awt.datatransfer](#)

[java.awt.dnd](#)

[java.awt.event](#)

[Java.awt.font](#)

[java.awt.geom](#)

[java.awt.im](#)

[java.awt.image](#)

[java.awt.image.renderable](#)

[java.awt.peer \(nur Java 1.1\)](#)

[java.awt.print](#)

[java.beans](#)

[java.beans.beancontext](#)

[java.io](#)

[java.lang](#)

[java.lang.ref](#)
[java.lang.reflect](#)
[java.math](#)
[java.net](#)
[java.rmi](#)
[java.rmi.activation](#)
[java.rmi.dgc](#)
[java.rmi.registry](#)
[java.rmi.server](#)
[java.security](#)
[java.security.acl](#)
[java.security.cert](#)
[java.security.interfaces](#)
[java.security.spec](#)
[java.sql](#)
[java.text](#)
[java.util](#)
[java.util.zip](#)
[javax.accessibility](#)
[javax.swing](#)
[javax.swing.event](#)
[javax.swing.undo](#)

Anhang C Java-Ressourcen im Internet

[Die Website zur amerikanischen Ausgabe des Buches](#)
[Andere Bücher](#)
[Die JavaSoft-Site](#)
[Andere Websites](#)
[Java Newsgroups](#)
[Berufschancen](#)

Anhang D Die Konfiguration des Java Development Kit

[Die Konfiguration unter Windows 95 und Windows NT](#)
[Die Konfiguration unter Unix](#)

Anhang E Texteditoren und das JDK

[Die Auswahl eines Texteditors](#)

[Dateitypen in Windows 95 registrieren](#)

[Anhang F Inhalt der CD-ROM zum Buch](#)

[Stichwortverzeichnis](#)

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 1

Tag 1

Eine Handvoll Java

Dies stellt das Ergebnis des fast 15 Jahre dauernden Versuchs dar, eine bessere Programmiersprache bzw. eine bessere Programmierumgebung zu schaffen, um einfachere und verlässlichere Software zu erzeugen.

Bill Joy, Mitbegründer von Sun Microsystems

Der verrückte Glöckner hatte recht. An einem Ort wie diesem gibt es Geld zu verdienen.

Der Mann ohne Namen aus »Für eine Hand voll Dollars«

Als die Programmiersprache Java im November 1995 das erste Mal der Öffentlichkeit präsentiert wurde, hatte dies viel Ähnlichkeit mit einem Revolverhelden à la Clint Eastwood, der in eine unfreundliche Westernstadt kommt.

Wie Clint war auch Java etwas, was die Bewohner der Stadt noch nie zuvor gesehen hatten. Java war eine Programmiersprache, die auf Webseiten lief und einen eigenen Platz neben Grafiken, Text, Audio und den allgegenwärtigen »Under construction«- Zeichen beanspruchte. Die Leute kamen von weit her ins Silicon Valley - meistens über Internet-Verbindungen, manche aber auch persönlich -, um einen Blick auf die Sprache zu werfen.

Die Legenden um Java überholten allerdings manchmal die Wirklichkeit ein wenig...

- »Java-Programme laufen ohne Modifikation reibungslos auf unterschiedlichen Computerplattformen!«
- »Java beendet Microsofts Hegemonie bei den Betriebssystemen!«
- »Java macht Computerbuchautoren zu international verehrten Berühmtheiten!«

In ähnlicher Weise haftete den Revolverhelden, die Clint spielte, schnell eine Legende an ...

- »Er ißt Kugeln zum Frühstück!«
- »Er schlägt so fest, daß Dein Großvater einen blauen Fleck bekommt!«
- »Er kann jemanden mit einem Blick töten!«

Java hat in den letzten drei Jahren viele der Erwartungen des anfänglichen Hypes erfüllt. Das Release 1.2 ist ein weiterer Beweis des anhaltenden Erfolges und Wachstums der Programmiersprache.

Beginnend mit dem heutigen Tag werden Sie viel über die Sprache Java lernen und warum sie so schnell in der Computerindustrie prominent wurde. Sie werden mit Java 1.2 - der aktuellsten Version - Applikationen erstellen, die sowohl auf Ihrem PC laufen, als auch über ein Netzwerk wie das Internet ausgeführt werden können. Außerdem werden Sie mit Java 1.0.2 Programme erstellen, die auf Webseiten ausgeführt werden. Dies war die Java-Version, mit der Java bekannt wurde. Sie wird auch heute noch von den meisten Browsern unterstützt.

Wenn Sie das Ende des Buches so bei Seite 17.228 erreicht haben, dann werden Sie wahrscheinlich eine weitere Gemeinsamkeit von Java und Clint Eastwood erkannt haben ...

Java ist cool.

Nicht in dem Sinne »aus der Dusche kommend fluchen: Wo zum Henker ist mein Handtuch? « und auch nicht im Sinne »Ich bin ein Rap-Star und kann 75 frauenfeindliche Bemerkungen in einer Minute murmeln«.

Java ist cool, weil es eine bemerkenswerte Programmiersprache ist, die es vielen Programmieren leichter macht, bemerkenswerte Dinge zu tun. Java ist cool, da es sichtweisenverändernde Konzepte wie die objektorientierte Programmierung verständlicher macht.

Wie das Wort »Salsa« ist Java cool, weil es einfach Spaß macht, das Wort laut auszusprechen. Java auszusprechen ist wesentlich schöner als »Visual Basic«, »C plus plus«, »Algol« oder »Mumps«.

(»Mumps!« zu sagen macht in gewisser Weise auch Spaß; gegenüber den anderen Begriffen ist Java allerdings wesentlich cooler.)

Wenn Sie sich durch die 21 Tage dieses Buches gearbeitet haben, werden Sie ein Experte für die gesamte Bandbreite der Möglichkeiten von Java sein. Dies schließt Grafik, Dateiein- und ausgaben, den Entwurf von Benutzerschnittstellen, die Ereignisbehandlung, die Datenbankprogrammierung und Animation mit ein. Sie werden Programme schreiben, die in Webseiten laufen, und andere, die auf Ihrem PC ausgeführt werden.

Die Ziele des heutigen Tages sind dagegen ziemlich bescheiden. Sie lernen etwas über die folgenden Themen:

- Was Java heute darstellt, und wie es dazu kam.
- Warum es sich lohnt, Java zu lernen, und warum Java ein würdiger Konkurrent zu anderen Programmiersprachen ist.
- Was Sie benötigen, um mit dem Schreiben von Java-Programmen zu beginnen - welche Software, welche Fähigkeiten und einige Grundbegriffe.
- Wie Sie Ihr erstes Java-Programm schreiben.

Was ist Java?

Ausgehend von dem riesigen Presserummel, den es um Java in den letzten paar Jahren gab, und der großen Zahl von Büchern zu Java (1.054 laut der aktuellsten Zählung von JavaWorld), werden Sie eventuell einen etwas übertriebenen Eindruck davon haben, was Java zu leisten imstande ist.

Java ist eine Programmiersprache, die sehr gut dafür ausgerüstet ist, Software zu entwerfen, die in Verbindung mit dem Internet läuft. Java ist aber auch eine objektorientierte Sprache, die von einer Methode Gebrauch macht, die in der Welt des Software- Design immer nützlicher wird. Außerdem ist Java eine plattformübergreifende Sprache. Das heißt, Java-Programme können so entwickelt werden, daß Sie in gleicher Weise auf Microsoft Windows, Apple Macintosh und Solaris-Maschinen - um nur einige zu nennen - laufen.

Java ist wesentlich näher an beliebten Programmiersprachen, wie z. B. C, C++, Visual Basic und Delphi, als HTML oder eine simple Skriptsprache wie JavaScript.

Lebensraum: Web

Java ist bekannt für seine Fähigkeit, in Webseiten zu laufen. Sowohl der Netscape Navigator als auch der Microsoft Internet Explorer können ein Java-Programm herunterladen und es lokal auf dem System des Benutzers ausführen.

Diese Programme, die Applets genannt werden, werden in Webseiten auf ähnliche Weise wie Bilder eingebunden. Im Gegensatz zu Bildern können Applets interaktiv sein - sie können Eingaben des Benutzers entgegennehmen, darauf reagieren und sich ständig verändernde Inhalte präsentieren.

Applets können zur Erzeugung von Animationen, Grafiken, Formularen, die sofort auf die Eingaben des Lesers reagieren, Spielen oder anderen interaktiven Effekte auf einer Webseite neben Text und Bildern verwendet werden. Abbildung 1.1 zeigt ein Applet, das im Netscape Navigator 4.04 läuft. Java wird hier dazu verwendet, einen Spieler gegen drei Gegner aus dem Computer-Domino zu spielen.



Das Domino-Applet wurde von Eric Carroll geschrieben, einem Java-Programmierer und Comic-Zeichner. Er ist auch der Gegner aus dem Computer mit der Igelfrisur in Abbildung 1.1. Sie finden das Domino-Applet und seine Homepage unter: [http:// www.void.org/~eric/domino.html](http://www.void.org/~eric/domino.html).

Applets werden über das World Wide Web heruntergeladen, wie das auch bei HTML- Seiten, Grafiken bzw. beliebigen anderen Elementen einer Website der Fall ist. In einem Browser, der für die Verarbeitung von Java ausgelegt ist, startet die Ausführung des Applets, nachdem es heruntergeladen ist.

Applets werden mit Java geschrieben, anschließend werden sie in eine Form kompiliert, die es erlaubt, das Programm auszuführen. Zu guter Letzt werden sie auf einem Web-Server abgelegt. Die meisten Web-Server sind in der Lage, Java-Dateien zu übertragen, ohne daß etwas an deren Konfiguration geändert werden müßte.



Fast alle Applets werden heute mit Java 1.0.2, der ersten verbreiteten Java-Version, geschrieben, da die meisten führenden Browser-Hersteller langsam bei der Integration von Java 1.1 und 1.2 sind. Netscape hat einen Softwarepatch veröffentlicht, der den Navigator um die Unterstützung von Java 1.1 erweitert. Microsoft hat hingegen keinerlei Pläne bezüglich der Java-1.1- Unterstützung im Internet Explorer 4 veröffentlicht.

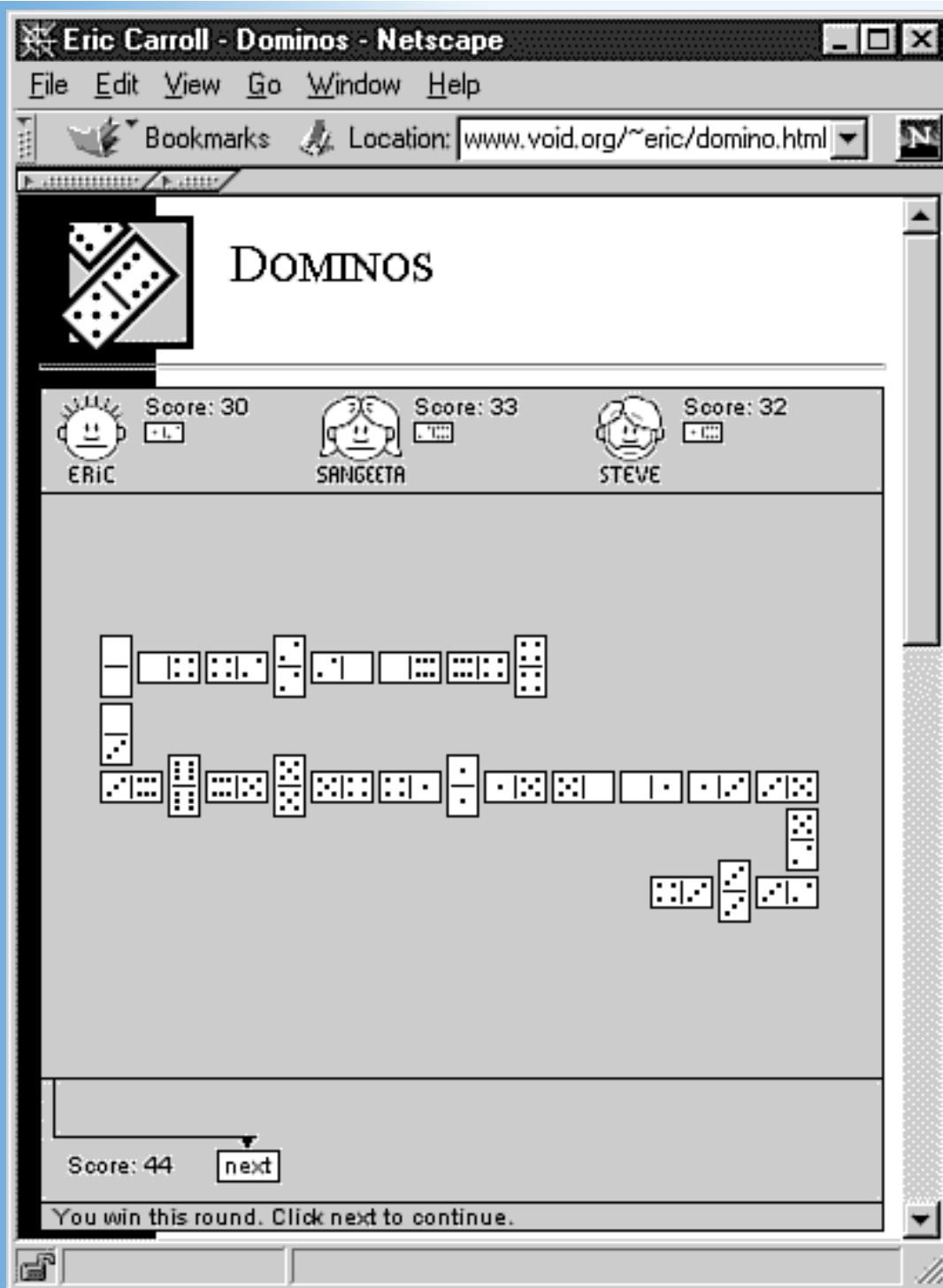


Abbildung 1.1:
Ein Applet im Netscape Navigator 4.04

Web-User mit einem Browser, der Java nicht unterstützt, sehen statt des Applets eventuell einen Text, eine Grafik oder gar nichts - abhängig davon, ob der Designer der Seite eine Alternative zu dem Java-Applet vorgesehen hat. Sie lernen im Laufe des Buches mehr über die Zusammenarbeit zwischen Applets, Browsern und dem World Wide Web.

Obwohl Applets die beliebteste Anwendung für Java darstellen, sind sie nur eine Möglichkeit, die Sprache zu verwenden. Wie bei Visual C++, Visual Basic und Delphi handelt es sich bei Java um eine robuste Sprache, mit der sich die unterschiedlichste Software entwickeln läßt. Diese Software kann grafische Benutzeroberflächen, Netzwerke, Datenbankverbindungen und andere sehr ausgefeilte Funktionalitäten unterstützen. Zur Unterscheidung von Applets werden Java-Programme, die nicht über das Web ausgeführt werden, Applikationen genannt.

Die inoffizielle Biographie von Java

Die Programmiersprache Java wurde 1991 bei Sun Microsystems als Teil des Green- Projekts entwickelt. Das Green-Projekt war eine Forschungsgruppe, die Software zur Steuerung von Konsumelektronikgeräten entwickelte. Die Wissenschaftler hofften, die Programmiersprache zu entwickeln, die die intelligenten Anwendungen der Zukunft steuert - interaktive Fernseher, interaktive Toaster usw. Suns Wissenschaftler wollten außerdem, daß die einzelnen Geräte miteinander kommunizieren können. Auf diese Weise sollte der Rasenmäher den Mixer anweisen können, Ihnen mitzuteilen, daß Ihre Nachbarn wieder zu Hause sind und nackt in der Sonne baden.

Um ihre Forschungen voranzutreiben, entwickelten die Green-Wissenschaftler einen Geräteprototypen mit dem Namen Star7 - ein ferngesteuertes Gerät, das in der Lage war, mit Angehörigen seiner Art zu kommunizieren. Die ursprüngliche Idee war es, das Betriebssystem des Star7 in C++ zu entwickeln, der extrem beliebten, von Bjarne Stroustrup entwickelten objektorientierten Programmiersprache. Allerdings hatte James Gosling vom Green-Projekt bald genug von C++ in bezug auf diese Aufgabenstellung. Aus diesem Grund verbarrikadierte er sich in seinem Büro und schrieb eine neue Programmiersprache, um den Star7 besser steuern zu können. Die Sprache nannte Gosling Oak - zu deutsch: Eiche - zu ehren eines Baumes, den er von seinem Bürofenster aus sehen konnte.



Sun stellte später fest, daß der Name Oak bereits vergeben war, hielt sich allerdings bei der Neubennennung der Sprache nicht an Goslings Blick-aus- dem-Fenster-Methode. Wäre dem so gewesen, hätten Sie vielleicht eine Sprache mit einem der folgenden Namen in 21 Tagen durcharbeiten müssen:

- Shrubbery (Gebüsch)
- OfficeBuildingNextDoor (Bürogebäude gegenüber)
- LightPole (Laterne)
- WindowWasher (Fensterwäscher)
- SecretaryLeavingForLunch (Sekretärin geht zum Mittagessen)
- WeirdSecurityGuard (Seltsamer Wachmann)
- FatGuyMowing (Dicker Mann beim Mähen)

Da bei der Entwicklung von Java andere Anwendungen als der Standard-PC im Vordergrund standen, mußte Java klein, effizient und leicht auf eine ganze Reihe von Hardware-Plattformen und -Geräten portierbar sein. Außerdem mußte Java zuverlässig sein. Man hat gelernt, mit den gelegentlichen Systemcrashes und Störungen einer 5-Mbyte-Applikation zu leben. Allerdings läßt sich über einen schlecht programmierten Toaster, dessen unangenehme Angewohnheit sich in Form einer Stichflamme äußert, nicht so einfach hinwegsehen.

Obwohl Java sich nicht als Entwicklungstool für die Steuerung von Geräten und interaktiven Fernsehern hervorgetan hat, erwiesen sich die Eigenschaften, die gut für den Star7 waren, auch als gut für das Web:

- Java war klein - dadurch ließen sich Programme schneller über das Web herunterladen.
- Java war sicher - dies hielt Hacker davon ab, Programme mit zerstörerischen Absichten über den Browser in das System des Benutzers zu bringen.
- Java war portabel - dies erlaubte es, daß es unter Windows, auf dem Macintosh und anderen Plattformen ohne Modifikation lief.

Zusätzlich konnte Java als allgemeine Programmiersprache für die Entwicklung von Programmen, die auf verschiedenen Plattformen ausgeführt werden können, verwendet werden.

Um das Potential von Java zu demonstrieren und das Forschungsprojekt vor der Zurückstellung zu bewahren, wurde 1994 ein Web-Browser entwickelt, der Java-Applets ausführen konnte. Der Browser zeigte zwei elementare Dinge von Java: Zum einen die Möglichkeiten in bezug auf das World Wide Web und zum anderen, welche Programme mit Java erstellt werden können. Die Programmierer Patrick Naughton und Jonathan Payne entwickelten den Browser, der ursprünglich WebRunner hieß, aber dann in HotJava umgetauft wurde.

Obwohl Java und der Browser HotJava sehr viel Aufmerksamkeit in der Web-Gemeinde auf sich zogen, trat die Sprache ihren eigentlichen Höhenflug erst an, als Netscape als erste Firma im August 1995 die Sprache lizenzierte. Der Netscape-Chef und Jungmillionär Marc Andreessen war einer der ersten außerhalb von Sun, die das Potential von Java erkannten. Auf der JavaOne-Konferenz im Mai 1996 war er ein großer Befürworter der Sprache. »Java ist eine große Chance für uns alle«, teilte er den Anwesenden mit. Kurz nach Javas erstem öffentlichen Release gliederte Sun seine Java-Entwicklung in eine neue Tochtergesellschaft mit dem Namen JavaSoft aus. Hunderte von Mitarbeitern wurden bereitgestellt, um die Sprache weiter voranzubringen.

Die verschiedenen Versionen der Sprache

JavaSoft hat bisher drei Hauptversionen der Sprache Java veröffentlicht:

- Java 1.0.2 - diese Version wird von den meisten Web-Browsern unterstützt.
- Java 1.1.5 - das Release vom Frühling 1997. Es beinhaltete Verbesserungen im Bereich der Benutzerschnittstellen bei der Ereignisbehandlung und war konsistenter innerhalb der Sprache.
- Java 1.2 - die neue Version. Diese wurde für den öffentlichen Betatest zum ersten Mal im Dezember 1997 veröffentlicht.

Die Versionsnummern von Java entsprechen immer der Versionsnummer von Suns primärem Java-Entwicklungstool - dem Java Developer's Kit. Im allgemeinen wird es als JDK bezeichnet. Momentan ist das Kit in den Versionen 1.0.2, 1.1.5 und 1.2 verfügbar.



In Verbindung mit dem JDK 1.2 wurde auch der Begriff »Java 2« eingeführt. Die exakte neue Bezeichnung für das bisherige JDK 1.2 ist jetzt Java 2 SDK v1.2. Der Einfachheit halber soll aber in diesem Buch weiterhin vom JDK 1.2 die Rede sein. Nähere Informationen zu den genauen Bezeichnungen in Verbindung mit »Java 2« finden Sie auf der Webseite:

<http://java.sun.com/products/jdk/1.2/java2.html>

Das JDK war immer schon kostenlos auf der Website von JavaSoft unter <http://java.sun.com> verfügbar. Diese Verfügbarkeit ist einer der Faktoren für das rapide Wachstum von Java. Es ist immer das erste Entwicklungstool, das eine neue Java-Version unterstützt, wenn diese veröffentlicht wird - oftmals sechs Monate, bevor andere Entwicklungswerkzeuge diese Version unterstützen.

Neben dem JDK gibt es mehr als ein Dutzend kommerzieller Java-Entwicklungstools für Java-Programmierer. Drunter die folgenden:

- Symantec Visual Café
- Borland JBuilder
- SuperCede
- Rogue Wave JFactory
- Natural Intelligence Roaster
- SunSoft Java WorkShop

Momentan ist das JDK immer noch das einzige Tool, das die Version 1.2 der Sprache voll unterstützt.



Die Programme in diesem Buch wurden mit dem JDK 1.2 Release Version, der aktuellsten Version des JDK, die beim Schreiben des Buches verfügbar war, getestet. Wenn Sie etwas anderes als das JDK beim Durcharbeiten des Buches verwenden, sollten Sie als erstes sicherstellen, daß es das JDK 1.2 voll unterstützt.

Spieglein, Spieglein an der Wand ...

Jeder, der die genaue Zukunft von Java kennt, sollte sich lieber um Venture-Kapital bemühen, als ein Buch zu schreiben. Die Technologie-Firma Kleiner, Perkins, Caufield and Byers (KPCB) hat \$100 Millionen bereitgestellt, um Start-up-Firmen zu unterstützen, deren Betätigungsfeld Java-bezogene Themen sind.

Die Erweiterungen in Java 1.2 bereiten Java gut auf seine Zukunft als ausgefeilte Sprache für unterschiedlichste Anwendungsgebiete vor. Frühe Versionen von Java waren besser für kleine webbasierte Software als für vollwertige Applikationen, wie Groupware-Anwendungen, Office-Suites und Netzwerk-Multiplayer-Spiele geeignet.

Das kann man von Java um 1.2 herum nicht mehr behaupten. Die folgende Liste beschreibt einige der fortgeschritteneren Features, die mit der aktuellen Version eingeführt wurden:

- Standardmäßige Unterstützung von Servlets - Java-Programmen, die auf einem Web-Server laufen, um einer Site bestimmte Funktionalitäten hinzuzufügen.
- Swing: Neue Features für die Erstellung grafischer Benutzeroberflächen sowohl im Stil bestimmter Betriebssysteme als auch in einem neuen Java-Look&Feel.
- Drag&Drop: Die Fähigkeit, Informationen interaktiv über verschiedene Applikationen hinweg bzw. von einem Teil der Benutzerschnittstelle eines Programms zu einem anderen Teil zu übertragen.
- Komplette Überarbeitung der Audio-Features von Java, was diese den Soundfähigkeiten anderer Sprachen annähert.

Sie bekommen in den nächsten drei Wochen die Möglichkeit, mit diesen und anderen neuen Features zu arbeiten.

Warum lesen Sie dieses Buch?

Früher war es einfacher, auszumachen, warum Leute ein Buch dieser Art wählten. Die meisten Leser wollten Java für die Erstellung von Applets verwenden.

Heute ist das nicht mehr so klar. Jede neue Version von Java führt Features ein, die Java über seine Wurzeln als interessante Web-Technologie hinaus erweitern.

Allerdings bleiben die Stärken von Java bestehen: Plattformunabhängigkeit, Objektorientierung und die einfache Erlernbarkeit.

Java ist plattformunabhängig

Plattformunabhängigkeit - die Fähigkeit, daß ein und dasselbe Programm auf unterschiedlichen Plattformen und unter verschiedenen Betriebssystemen läuft - ist einer der bedeutendsten Vorteile, die Java gegenüber anderen Programmiersprachen zu bieten hat.

Wenn Sie z.B. ein Programm, in C oder einer der meisten anderen Programmiersprachen kompilieren, dann übersetzt der Compiler Ihre Quelldateien in Maschinensprache - Befehle, die für den Prozessor in Ihrem System spezifisch sind. Wenn Sie Ihren Code auf einer Maschine mit Intel-Prozessor kompilieren, dann wird das Programm auf anderen Maschinen mit Intel-Prozessor laufen, auf Macs, Commodore VIC-20 oder anderen Maschinen dagegen nicht. Wenn Sie dasselbe Programm auf einer anderen Plattform verwenden wollen, müssen Sie Ihren Quellcode auf diese Plattform transferieren und dort neu kompilieren, um den für dieses System spezifischen Maschinencode zu erhalten. In vielen Fällen sind, auf Grund von Unterschieden innerhalb der Prozessoren und anderer Faktoren, Änderungen an dem Quellcode des Programms nötig, bevor es sich auf der neuen Maschine kompilieren läßt.

Die Abbildung 1.2 zeigt das Ergebnis eines plattformabhängigen Systems: Viele ausführbare Programme müssen für viele Systeme erzeugt werden.

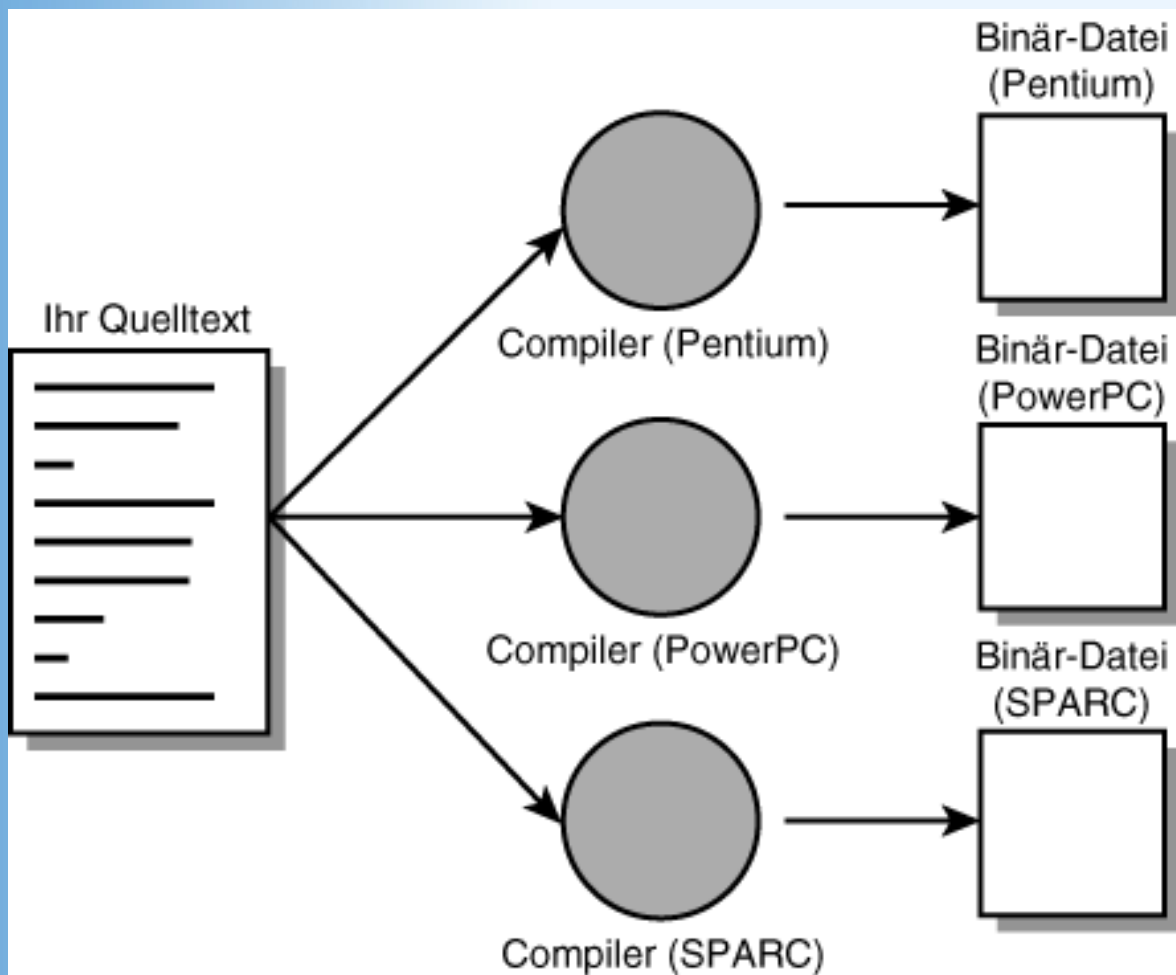


Abbildung 1.2:
Herkömmlich kompilierte Programme

Java-Programme erreichen die Plattformunabhängigkeit über eine virtuelle Maschine - eine Art Computer im Computer. Die virtuelle Maschine nimmt das Java-Programm und konvertiert die Anweisungen darin in Kommandos, die das jeweilige Betriebssystem verarbeiten kann. Dasselbe kompilierte Programm, das in einem Format namens Bytecode vorliegt, kann so auf jeder beliebigen Plattform bzw. unter jedem beliebigen Betriebssystem ausgeführt werden, das über eine Java Virtual Machine (die amerikanische Bezeichnung, Abk.: JVM)

Die virtuelle Maschine wird auch als Java Interpreter oder Java Runtime Environment (Java-Laufzeitumgebung) bezeichnet.

Wenn es Ihnen Probleme bereitet, die Rolle der virtuellen Maschine zu verstehen, dann hilft vielleicht die folgende Metapher: In der Originalserie von *Star Trek* (in Deutschland *Raumschiff Enterprise*) gab es ein Gerät, das Englisch (bzw. Deutsch) in die Sprache der Außerirdischen übersetzte, wenn sich diese mit der Crew der Enterprise unterhielten. Captain James T. Kirk mußte nicht bei jeder Landung auf einem neuen Planeten eine neue Sprache lernen, da der Universalübersetzer seine Worte so umsetzte, daß die Aliens ihn verstehen konnten. In gleicher Weise müssen Java-Programmierer nicht verschiedene Versionen eines Programms für jede Plattform erstellen, auf der dieses landet, da die virtuelle Maschine sich um die notwendige Übersetzung kümmert. (Natürlich setzte Kirk den Übersetzer dazu ein, mit den Frauen der anderen Welten zu flirten. Wir können allerdings weder ausdrücklich noch implizit eine Garantie dafür geben, daß Sie über Java eine Verabredung bekommen.)

Java ist auch auf Quellebene plattformunabhängig. Java-Programme werden vor der Kompilierung als Textdateien gespeichert. Diese Dateien können auf jeder Plattform erzeugt werden, die Java

unterstützt. Sie könnten z.B. ein Java-Programm auf einem Mac schreiben und dieses anschließend unter Windows 95 kompilieren.

Bytecode ähnelt dem Maschinencode, der von anderen Sprachen erzeugt wird. Allerdings ist dieser nicht für einen bestimmten Prozessor spezifisch. Er führt eine zusätzliche Schicht zwischen dem Quellcode und dem Maschinencode ein (siehe auch Abbildung 1.3).

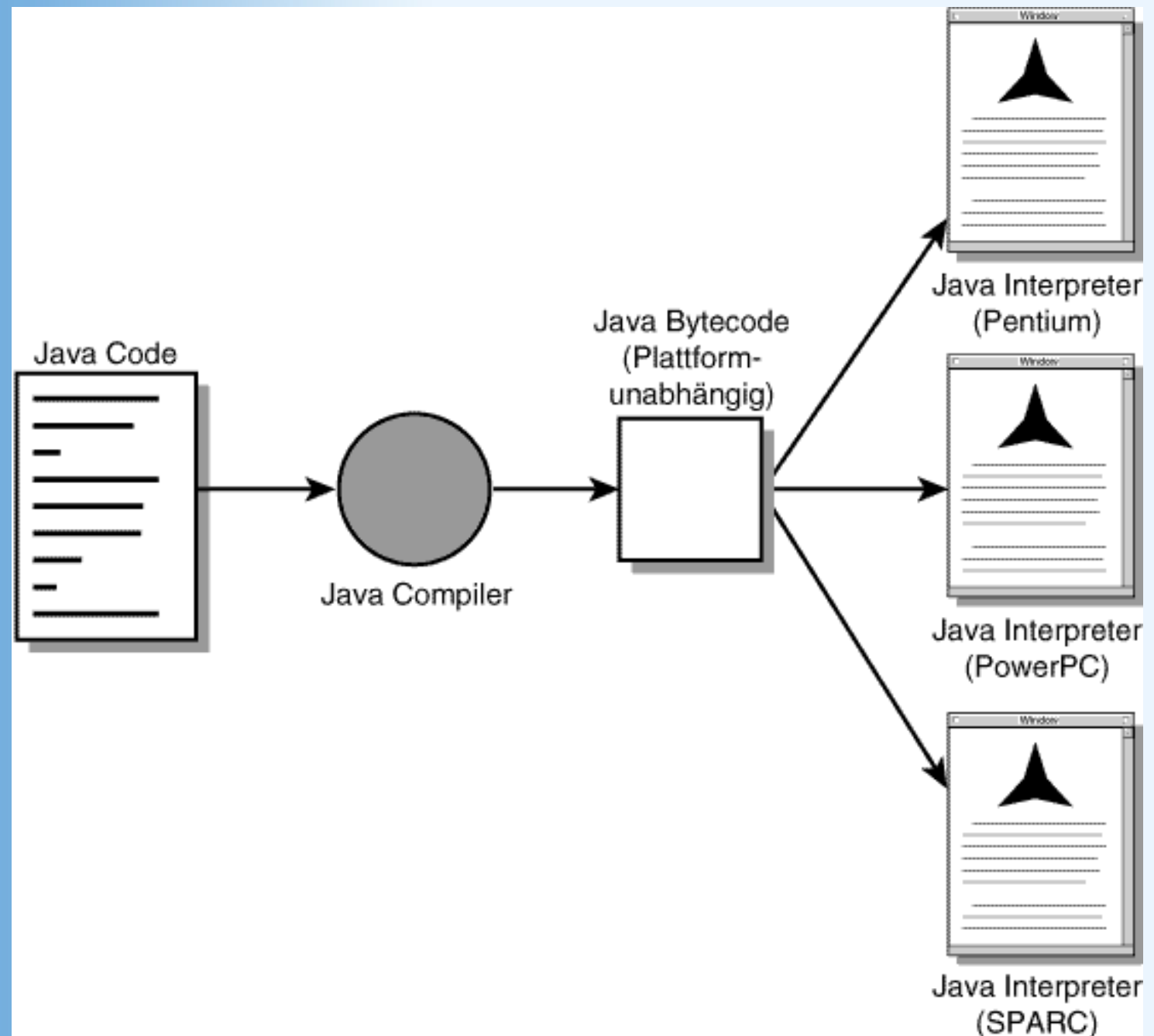


Abbildung 1.3:
Java erzeugt Multiplattform-Programme

Die Java Virtual Machine kann sich an unterschiedlichen Orten befinden. Für die Ausführung von Applets ist die virtuelle Maschine in die einzelnen Browser integriert, die Java unterstützen. Deshalb müssen Sie sich keine Gedanken darüber machen, ob sich die virtuelle Maschine auf dem System des Benutzers befindet.



So ganz sorglos kann man die Sache allerdings dann doch nicht angehen. Sie müssen sich Gedanken darüber machen, welche Java Virtual Machine der Browser unterstützt. Wenn Sie ein Applet erstellen, das neue Features von Java 1.2 verwendet, dann wird der Bytecode dieses Applets in einem Browser, der lediglich die virtuelle Maschine von Java 1.0.2 unterstützt, nicht funktionieren. Java Plug-In, eine Browsererweiterung, die von JavaSoft entwickelt wurde, ermöglicht es Entwicklern, eine andere virtuelle Maschine festzulegen, als eigentlich in dem Netscape Navigator oder dem Microsoft Internet Explorer integriert ist. Dies ermöglicht es Java-1.1- und 1.2- Applets, zu laufen, wenn die entsprechende virtuelle Maschine festgelegt wurde. Das JDK 1.2 enthält das Java Plug-In in der Version 1.2. Mehr Informationen über Java Plug-In finden Sie auf der folgenden Webseite:

<http://java.sun.com/products/plugin/index.html>

Java-Applikationen können dagegen nur auf Systemen ausgeführt werden, auf denen die entsprechende virtuelle Maschine installiert wurde. Wenn Sie Java-1.2-Applikationen auf Ihrem Computer ausführen wollen, müssen Sie als erstes die virtuelle Maschine installieren.

Wenn Sie an die Art gewöhnt sind, mit der Sprachen wie Visual Basic und Delphi plattformabhängigen Code erzeugen, dann werden Sie vielleicht denken, daß der Bytecode-Interpreter eine unnötige Schicht zwischen Ihrem Code und dem Maschinencode darstellt.

Dies wirft natürlich einige Performance-Fragen auf - Java-Programme werden langsamer ausgeführt als Programme von plattformabhängig kompilierten Sprachen wie C. Und der Geschwindigkeitsunterschied ist der wesentliche Kritikpunkt an Java. Manche Entwicklungstools beinhalten Just-In-Time-Compiler, die den Java-Bytecode mit größerer Geschwindigkeit ausführen können.

Die Möglichkeit, daß eine einzige Bytecode-Datei auf unterschiedlichsten Plattformen ausgeführt werden kann, ist entscheidend dafür, daß Java im World Wide Web funktioniert, da das Web ebenfalls plattformunabhängig ist.

Für viele einfache Java-Programme ist die Geschwindigkeit kein Thema. Wenn Sie Programme schreiben, die eine höhere Ausführungsgeschwindigkeit benötigen, als die virtuelle Maschine bietet, bieten sich mehrere Lösungen an:

- Sie können Aufrufe für systemspezifischen Maschinencode in Ihre Java-Programme integrieren. Dies macht ein Programm allerdings plattformabhängig.
- Verwenden Sie Just-In-Time-Compiler, die Java-Bytecode in systemspezifischen Code umwandeln.

Egal welche der beiden Lösungen Sie verwenden, Sie gewinnen Geschwindigkeit auf Kosten der Portabilität von Java. Eine Java-Applikation, die z. B. Windows-Funktionsaufrufe für den Zugriff auf die Festplatte verwendet, läuft ohne Änderung nicht auf einem Macintosh.

Java ist objektorientiert

Wenn Sie mit der objektorientierten Programmierung bis jetzt noch nicht vertraut sind, dann werden Sie in den nächsten sechs Tagen reichlich Gelegenheit erhalten, um dies zu ändern.

Als objektorientierte Programmierung - auch OOP genannt - wird eine Methode bezeichnet, bei der

Computerprogramme als eine Reihe von Objekten aufgebaut werden, die miteinander interagieren. Für andere ist es im wesentlichen eine Art, Programme zu organisieren. Jede Programmiersprache kann dazu verwendet werden, objektorientierte Programme zu erstellen.

Allerdings ziehen Sie den größten Nutzen aus der objektorientierten Programmierung, wenn Sie eine Sprache verwenden, die dafür entworfen wurde. Java erbte viele seiner OOP-Konzepte von C++, der Sprache, auf der Java zu einem großen Teil basiert. Java entleiht auch Konzepte aus anderen objektorientierten Sprachen.

Am Tag 2 lernen Sie mehr über objektorientierte Programmierung und Java.

Java ist leicht zu erlernen

Neben seiner Portabilität und der Objektorientierung ist Java kleiner und einfacher als andere vergleichbare Sprachen. Dies rührt von dem ursprünglichen Ziel für Java her, eine Sprache zu sein, die weniger Computermuskeln für die Ausführung benötigt - niemand wird \$3.000 für einen Pentium-II-Toaster mit MMX-Technologie ausgeben.

Java sollte einfacher zu schreiben, kompilieren, debuggen und zu lernen sein. Die Sprache wurde sehr stark nach dem Vorbild von C++ modelliert, und vieles von der Syntax und der objektorientierten Struktur kommt direkt von dieser Sprache. Wenn Sie C++- Programmierer sind, dann werden Sie in der Lage sein, Java wesentlich schneller zu lernen, und können einiges in der ersten Woche dieses Buches überspringen.

Trotz Javas Ähnlichkeiten mit C++ wurden die komplexesten und fehlerträchtigsten Aspekte der Sprache nicht in Java aufgenommen. Sie werden z. B. keine Zeiger oder Zeigerarithmetik in Java finden, da diese Features in einem Programm leicht zu Fehlern führen und sie schwerer als andere zu beheben sind. Strings und Arrays sind Java- Objekte, und das Speichermanagement wird von Java automatisiert, anstatt dies dem Programmierer zu übertragen. Erfahrene Programmierer werden diese Punkte vielleicht vermissen, wenn sie beginnen, mit Java zu programmieren, andere werden allerdings durch das Weglassen Java wesentlich schneller lernen.

Obwohl Java leichter zu lernen ist als viele andere Programmiersprachen, stellt Java für jemanden ohne Programmiererfahrung eine große Herausforderung dar. Der Umgang mit Java ist komplexer als der mit HTML oder etwa JavaScript - allerdings gibt es nichts, was ein Anfänger nicht meistern könnte.

Eintauchen in die Java-Programmierung

Lassen wir nun die Geschichtsstunde enden, und widmen wir uns der zweiten Hälfte des heutigen Tages. Sie erhalten Gelegenheit, Java in Aktion zu erleben, während Sie Ihr erstes Java-Programm - eine Applikation - erstellen.

Bevor Sie allerdings beginnen können, müssen Sie ein Java-1.2-Entwicklungstool auf Ihrem System installieren.

Ein Entwicklungstool auswählen

Um Java-Programme zu schreiben, benötigen Sie eine Entwicklungssoftware für Java. Falls Ihr System bereits in der Lage ist, Applets auszuführen, wenn Sie im Internet surfen, werden Sie vielleicht denken, daß damit schon alles für die Entwicklung vorbereitet ist. Dies ist allerdings nicht der Fall - Sie brauchen noch eine Entwicklungssoftware, um eigene Java-Programme zu erstellen und auszuführen.

Wenn Sie dieses Buch voll ausschöpfen wollen, benötigen Sie ein Entwicklungstool, das Java 1.2 komplett unterstützt. Momentan kommt nur das Java Developer's Kit (JDK) in Frage. Das JDK ist immer das erste Tool, das eine neue Java-Version unterstützt, und das JDK 1.2 Release Version ist, während ich dieses Buch schreibe, die aktuellste Version.

Das JDK besteht aus einer Reihe kommandozeilenorientierter Programme. Diese Programme sind rein textorientiert und besitzen keine grafische Benutzeroberfläche. Als Programmierer führt man die einzelnen Programme aus, indem man an der Eingabeaufforderung Befehle wie den folgenden eingibt:

```
java GetFunky.class
```

Dieses Kommando weist das Programm `java` - den Bytecode-Interpreter - an, eine Datei mit dem Namen `GetFunky.class` auszuführen. (Wie Sie noch zu einem späteren Zeitpunkt am heutigen Tag sehen werden, tragen alle kompilierten Java-Programme die Erweiterung `.class`.)

Anwender von Windows 95 müssen die MS-DOS-Eingabeaufforderung verwenden (Start | Programme | MS-DOS-Eingabeaufforderung in der Taskleiste), um ein Fenster zu öffnen, in das Befehle eingegeben werden können.

Dies ist natürlich weit entfernt von den modernen Entwicklungstools, die eine grafische Benutzeroberfläche, Debugger, Quellcode-Editoren und andere nette Details bieten.

Wenn Sie ein anderes Java-Entwicklungstool haben und sicher sind, daß es Java 1.2 unterstützt, dann können Sie es verwenden, um die Programme in diesem Buch zu erstellen. Andernfalls sollten Sie das JDK verwenden.

Die Installation des JDK

Die Version 1.2 des Java Developer's Kit ist momentan für die folgenden Plattformen verfügbar:

- Windows 95
- Windows NT
- Solaris SPARC
- Solaris x86

Wenn Sie auf einer anderen Plattform arbeiten, wie z. B. dem Apple Macintosh, dann können Sie durch einen Besuch der offiziellen Java-Site von Sun (<http://java.sun.com>) feststellen, ob es dort inzwischen eine 1.2-Version des JDK für diese Plattform gibt. Aktuell finden Sie eine Liste aller bekannten Versionen des JDK für die verschiedenen Plattformen auf der folgenden Seite:

<http://java.sun.com:80/cgi-bin/java-ports.cgi>

Wenn Sie das JDK von einer CD mit Varianten für verschiedene Betriebssysteme installieren, müssen Sie die Datei entsprechend Ihrer Computerplattform auswählen:

- Windows-95- und Windows-NT-Anwender: Die Installationsdatei `jdk12-win32.exe` befindet sich im Verzeichnis `\JDK\Win95nt4`.
- Solaris-SPARC-Anwender: Die Installationsdatei `jdk12-solaris2-sparc.bin` befindet sich im Verzeichnis `\JDK\Sparcsol`.
- Solaris-x86-Anwender: Die Installationsdatei `jdk12-solaris2-x86.bin` befindet sich im Verzeichnis `\JDK\Intelsol`.

Wenn Sie keinen Zugriff auf ein CD-ROM-Laufwerk haben, dann können Sie sich das JDK auch aus

dem World Wide Web herunterladen. Suns offizielle Windows- und Solaris-Versionen finden Sie auf der folgenden Webseite:

<http://www.javasoft.com/products/JDK/1.2/index.html>

Diese Seite beinhaltet Informationen zur Installation und einen Link, um das JDK für Ihre Plattform herunterzuladen und es in einem Ordner auf Ihrem System zu speichern.

Nachdem Sie die Datei heruntergeladen haben, sollten Sie prüfen, ob die gesamte Datei heruntergeladen wurde. Bei den Installationsanweisungen im Web ist auch die Größe der JDK-Datei für Ihre Plattform aufgeführt.

Um die Größe der Datei unter Windows 95 oder NT zu prüfen, wechseln Sie zu dem Ordner, der die Datei enthält, und klicken mit der rechten Maustaste auf die Datei. Es erscheint ein Kontextmenü. Wählen Sie in diesem Menü den Eintrag Eigenschaften. In dem Dialog, der sich öffnet, wird neben anderen relevanten Informationen die Größe der Datei angezeigt.

Die Installation unter Windows 95 und NT

Bevor Sie das JDK auf Ihrem System installieren, sollten Sie sicherstellen, daß keine anderen Java-Entwicklungstools installiert sind. Wenn mehr als ein Java-Entwicklungstool installiert ist, kann das zu Konfigurationsproblemen führen, wenn Sie versuchen, das JDK zu verwenden.

Um das JDK unter Windows 95 oder NT zu installieren, klicken Sie doppelt auf die Installationsdatei oder verwenden das Kommando Start | Ausführen, um die Datei zu lokalisieren und auszuführen.

Im Anschluß an den Dialog, der Sie fragt, ob Sie das JDK 1.2 installieren wollen, wird der Installationsassistent (siehe auch Abbildung 1.4) angezeigt. Sie können in diesem Fenster festlegen, wie das JDK auf Ihrem System installiert werden soll.

Die Standardeinstellungen in diesem Assistenten sollten in der Regel für die meisten Anwender zutreffend sein. Das JDK wird im Ordner \jdk1.2 auf Ihrem C:-Laufwerk installiert, solange Sie nicht auf die Schaltfläche Browse... klicken, um einen anderen Ordner zu wählen.



Probleme, die Sie eventuell mit der Konfiguration des JDK haben, lassen sich leichter beheben, wenn Sie den Standardordner \jdk1.2 verwenden.

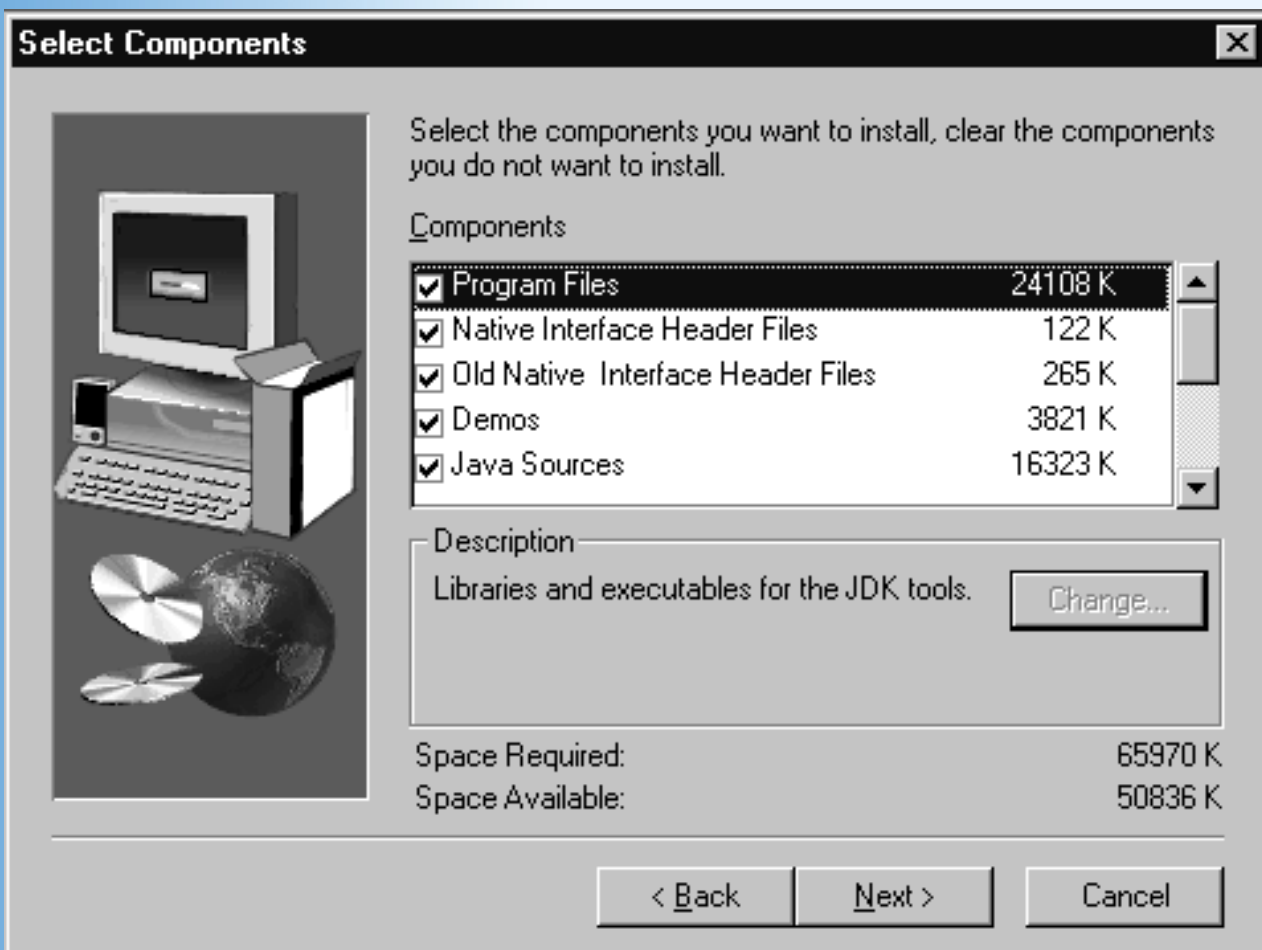


Abbildung 1.4:
Der Installationsassistent des JDK

Der Assistent installiert drei Komponenten des JDK:

- **Programmdateien:** Die ausführbaren Dateien, die benötigt werden, um Ihre Java- Projekte zu erstellen, zu kompilieren und zu testen.
- **Bibliotheks- und Header-Dateien:** Dateien, die nur von Programmierern verwendet werden, die nativen Code aus ihren Java-Programmen heraus aufrufen. Diese sind für die Tutorials in diesem Buch nicht wichtig.
- **Demodateien:** Java-1.2-Programme, die sowohl in ausführbarer Form als auch im Quelltext vorliegen. Den Quelltext können Sie durcharbeiten, um mehr über die Sprache zu lernen.

Wenn Sie die Standardinstallation akzeptieren, benötigen Sie 23 Mbyte auf Ihrer Festplatte. Falls Sie alle Optionen außer den Programmdateien weglassen, spart Ihnen das 2,4 Mbyte - für den Fall, daß Sie Platz sparen müssen, um das JDK auf eine überfüllte Platte zu quetschen.

Zusätzlich können Sie die Quelldateien installieren - der Quelltext der Sprache Java selbst ist öffentlich verfügbar. Allerdings benötigen diese Dateien mehr als 11 Mbyte und werden in diesem Buch nicht benötigt.



Nachdem Sie das JDK installiert haben, werden Sie vielleicht bemerken, daß zwei der installierten Dateien in den Unterordnern JDK\lib bzw. JDK\jre\lib den Namen tools.jar bzw. rt.jar tragen. Obwohl

diese Dateien JAR-Archive sind, sollten Sie sie nicht entpacken. Das JDK kann sie in diesen Ordnern in dem Archiv-Format lesen.

Die Installation unter Solaris

Die Solaris-Version des JDK von Sun kann auf den folgenden Plattformen installiert werden:

- SPARC-Systemen mit Solaris 2.4 oder höher
- x86-Systemen mit Solaris 2.5 oder höher

Das JDK-Installationsarchiv sollte in ein Verzeichnis entpackt werden, in dem noch kein Unterverzeichnis mit dem Namen `jdk1.2` existiert, da Sie ansonsten bereits vorhandene Dateien auf Ihrem System überschreiben.

Wenn Sie die Installationsdatei heruntergeladen haben, sollten Sie sicherstellen, daß Sie auf die Datei über das Shell-Kommando `chmod a+x` korrekt zugreifen können.

SPARC-Anwender würden z. B. das folgende Kommando verwenden:

```
% chmod a+x jdk12-solaris2-sparc.bin
```

Um das JDK zu installieren, nachdem Sie `chmod` ausgeführt haben, verwenden Sie das Shell-Fenster und geben das Kommando `./` gefolgt von dem Namen der Archivdatei ein, wie das im folgenden der Fall ist:

```
% ./jdk12-solaris2-sparc.bin
```

Die Installation testen

In einer idealen Welt sollte das JDK nach der Installation richtig funktionieren. Käse sollte fettfrei sein, Präsidenten sollten tugendhaft sein, und Jimmy Johnson sollte der Trainer der Dallas Cowboys sein.

Die größten Probleme beim Erlernen von Java ergeben sich aus Fehlern bei der Konfiguration des JDK.

Windows-Anwender können Ihre Installation des JDK testen, indem Sie die MS-DOS-Eingabeaufforderung verwenden (Start | Programme | MS-DOS-Eingabeaufforderung in der Taskleiste). Dies öffnet ein Fenster, in dem Sie Befehle unter MS-DOS (dem Betriebssystem, das Windows 3.1 vorausging) eingeben können.

Die MS-DOS-Eingabeaufforderung wird auch einfach Eingabeaufforderung, Kommandozeile oder Prompt genannt. Diese Namen sollen zum Ausdruck bringen, daß Sie hier Befehle eingeben können, die das Betriebssystem ausführt.



MS-DOS kann für Leute, die an grafische Benutzeroberflächen wie Windows 95 gewöhnt sind, einschüchternd sein. Allerdings können Sie das JDK, ohne ein bißchen MS-DOS zu lernen, nicht einsetzen. Dieses Buch wird Ihnen diverse Tips geben, damit Sie so wenig MS-DOS wie möglich lernen müssen.

Um zu testen, ob Ihr System die richtige Version des JDK finden kann, geben Sie an der Eingabeaufforderung das folgende Kommando ein:

```
java -version
```

Sie sollten als Reaktion die folgende Meldung erhalten:

```
java version "1.2"
```

Wenn Sie die falsche Versionsnummer an dieser Stelle sehen oder die Meldung Befehl oder Dateiname nicht gefunden, dann kann Ihr System die richtige Version der Datei java.exe (die Datei, die Java-Programme ausführt) nicht finden.

Dieser Fehler muß korrigiert werden, bevor Sie damit beginnen, Java-Programme zu schreiben. In Anhang D finden Sie Informationen zur Konfiguration des JDK.

Die erste Java-Applikation

Lassen Sie uns nun mit der eigentlichen Arbeit beginnen. Starten Sie mit einer Applikation: ein Programm, das mit das Seltsamste, was einer Berühmtheit von jemandem aus der Menge entgegengebrüllt wurde, auf dem Bildschirm anzeigt: What's the frequency, Kenneth?



Am 4. Oktober 1986 schrie ein Mann »What's the frequency, Kenneth?« Fernsehkoordinator Dan Rather entgegen, kurz bevor er ihn auf einer öffentlichen Straße in New York City zusammenschlug. Jahrelang verstand niemand das Motiv für diesen Angriff, und die Pop-Gruppe R.E.M. machte diesen Ausspruch in einem Song unsterblich. Der Mann wurde später verhaftet, nachdem er 1994 einen Techniker von NBC niedergeschossen hatte. Er erzählte einem Psychiater, daß die TV-Sender ihn verfolgen würden.

Java-Applikationen sind alleinstehende Programme, die keinen Web-Browser zur Ausführung benötigen. Sie sind fast wie die Programme, die Sie hauptsächlich auf Ihrem System verwenden: Sie starten diese lokal über die Maus oder über einen Befehl an der Eingabeaufforderung.

Obwohl ein Java-Programm so entworfen werden kann, daß es sowohl ein Applet als auch eine Applikation ist, werden die meisten Programme, auf die Sie stoßen, entweder das eine oder das andere sein.

In der gesamten ersten Woche, in der Sie die Sprache Java lernen, werden Sie Applikationen schreiben. Dieses Wissen wird in der zweiten Woche um die Applet-Programmierung erweitert. Wenn Sie einer der Leser sein sollten, die nur an der Applet- Programmierung interessiert sind, sollten Sie trotzdem nicht versuchen, direkt zur zweiten Woche zu springen. Alles, was Sie bei der Erstellung einfacher Java-Applikationen lernen, ist ebenso für die Erstellung von Applets gültig, und bei den Grundlagen zu beginnen, ist das beste. Sie werden in den Tagen 8 bis 14 eine ganze Menge Applets erstellen.

Die Erstellung des Quellcodes

Wie das bei den meisten Programmiersprachen der Fall ist, werden Ihre Java-Quelldateien als reine Textdateien gespeichert. Sie können diese mit jedem beliebigen Editor oder Textverarbeitungsprogramm erstellen, der/das reinen Text speichern kann - einem Format, das auch ASCII- oder DOS-Text genannt wird. Windows-95-Anwender können Java-Programme mit Notepad, DOS Edit bzw. WordPad oder WinWord erstellen - vorausgesetzt sie achten darauf, daß sie die

Dateien im Text-Format abspeichern anstatt in dem jeweiligen WinWord-Format. Unix-Anwender können Programme mit emacs, pico und vi erstellen. Den Mac-Anwendern steht SimpleText für die Erstellung von Java-Quelldateien zur Verfügung.

Das Java Developer's Kit beinhaltet keinen Texteditor. Die meisten anderen Entwicklungstools verfügen aber über Ihren eigenen Editor, um die Quelltexte zu erstellen.

Wenn Sie Windows 95 oder NT verwenden, dann fügt ein Texteditor wie Notepad eventuell .txt als zusätzliche Erweiterung an den Dateinamen einer Java-Datei an, die Sie speichern. Dies hätte zur Folge, daß z. B. aus dem Dateinamen GetFunky.java GetFunky.java.txt wird. Um dieses Problem zu vermeiden, sollten Sie den Dateinamen beim Speichern in Anführungszeichen einschließen - Abbildung 1.5 zeigt diese Technik beim Speichern der Quelltext-Datei »Craps.java« mit Notepad von Windows.

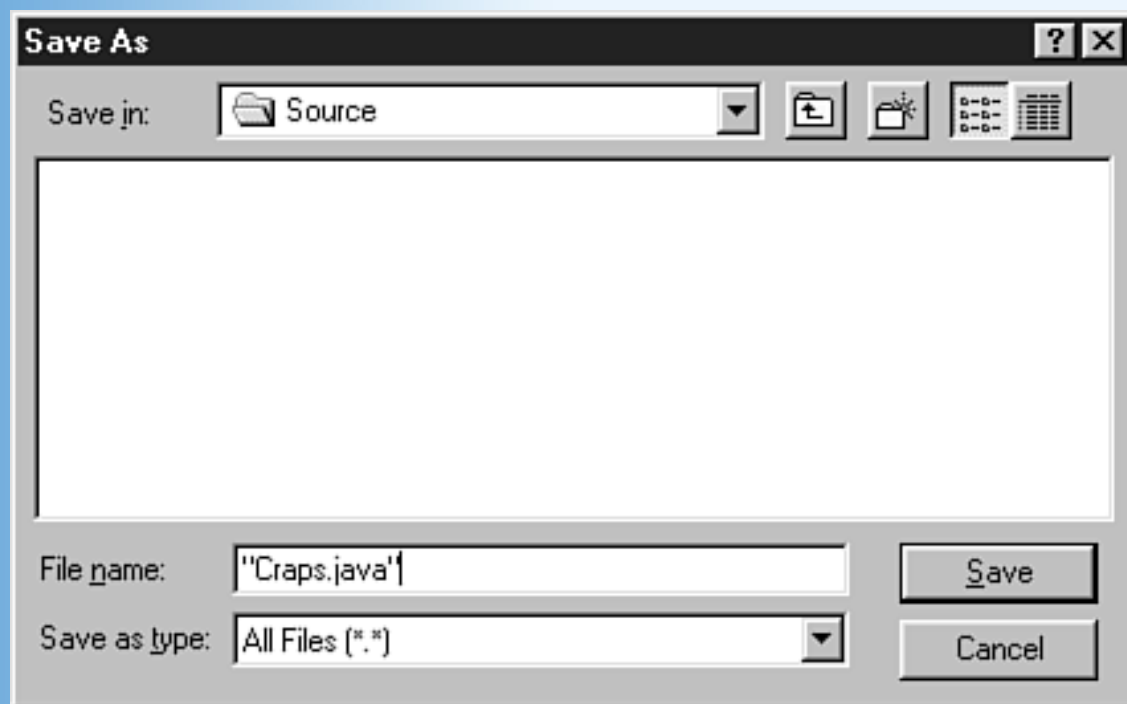


Abbildung 1.5:
Das Speichern einer Quelldatei



Eine wesentlich bessere Lösung ist es, die .java-Dateien fest mit dem Texteditor, den Sie verwenden, zu verknüpfen. Dies ermöglicht es Ihnen über einen Doppelklick auf den Namen, eine Quelltextdatei in einem Ordner zur Bearbeitung zu öffnen. Wie Sie eine solche Verknüpfung einrichten, erfahren Sie in Anhang E.

Das Programm schreiben

Starten Sie den Editor Ihrer Wahl, und geben Sie das Java-Programm ein, das in Listing 1.1 aufgeführt ist. Achten Sie darauf, daß Sie alle geschweiften Klammern, eckigen Klammern und Anführungszeichen genauso wie in dem Listing eingeben. Achten Sie außerdem darauf, daß Sie die Groß- und Kleinschreibung richtig aus dem Text übernehmen.

Listing 1.1: Der Quellcode der HalloDan-Applikation

```

1: class HelloDan {
2:     public static void main (String[] arguments) {
3:         System.out.println("What's the frequency, Kenneth?");
4:     }
5: }

```

Die Zeilennummern und die Doppelpunkte an der linken Seite im Listing 1.1 sind nicht Teil des Programms - sie wurden eingefügt, damit man sich im Buch auf bestimmte Zeilen in einem Programm beziehen kann. Sollten Sie sich bei einem Quelltext in diesem Buch einmal nicht ganz sicher sein, dann können Sie diesen mit der entsprechenden Kopie auf der CD des Buches vergleichen.

Nachdem Sie das Programm abgetippt haben, sichern Sie die Datei irgendwo auf Ihrer Platte unter dem Namen HelloDan.java.



Wenn Sie Anwender des JDK unter Windows 95 sind und versuchen, so wenig wie irgendetwas über MS-DOS zu lernen, dann öffnen Sie auf Ihrem System den Stammordner und legen einen neuen Unterordner mit dem Namen J21Work an. Speichern Sie HelloDan.java und alle anderen Java-Quelldateien aus diesem Buch in dieses Verzeichnis. Sie werden bald verstehen, warum.

Dieser Dateiname ist sehr wichtig. Java-Quelldateien müssen mit der Erweiterung .java gesichert werden, und der Name muß demjenigen in dem Haupt-class-Statement in der Quelldatei entsprechen. Die Groß-/Kleinschreibung spielt ebenfalls eine wesentliche Rolle, diese muß nämlich ebenfalls übereinstimmen.

Java-Quelldateien werden in Bytecode kompiliert. Die entstehenden Dateien tragen die Erweiterung .class. In manchen Beziehungen ist der Begriff *Klasse* synonym mit *Programm* (in den nächsten drei Tagen werden Sie mehr über Klassen lernen).

In der zweiten Zeile von Listing 1.1 wird festgelegt, daß das Java-Programm aus der Klasse HelloDan besteht, was wiederum bedeutet, daß der Dateiname HelloDan.java sein muß. Wenn Sie Ihre Quelldatei anders benennen (dazu zählt sogar hellodan.java oder Hellodan.java), werden Sie nicht in der Lage sein, das Programm zu kompilieren.

Kompilierung und Ausführung unter Windows

Jetzt sind Sie bereit, die Datei zu kompilieren. Wenn Sie ein anderes Entwicklungswerkzeug als das JDK verwenden, sollten Sie in der Dokumentation der Software nachsehen, wie die Kompilierung von Java-Programmen im Detail funktioniert. Es wird wahrscheinlich eine sehr einfache Operation sein, wie z. B. ein Klick auf eine Schaltfläche oder ein Kommando in einem Menü.

Beim JDK verwenden Sie das Kommandozeilen-Tool javac, den Java-Compiler. Der Compiler liest eine .java-Quelldatei und erzeugt ein oder mehrere .class-Dateien, die von der Java Virtual Machine ausgeführt werden können.

Windows-95-Anwender sollten die MS-DOS-Eingabeaufforderung starten (Start | Programme | MS-DOS-Eingabeaufforderung) und in den Ordner wechseln, der HelloDan.java enthält.

Wenn Sie die Datei in den neu erstellten Ordner J21Work direkt unterhalb des Stammverzeichnisses

gespeichert haben, dann ist dafür das folgende MS-DOS-Kommando nötig:

```
cd \J21Work
```

cd ist die Abkürzung für »change Directory« (engl. »Wechsle das Verzeichnis«) - die Begriffe *Ordner* und *Verzeichnis* sind gleichbedeutend.

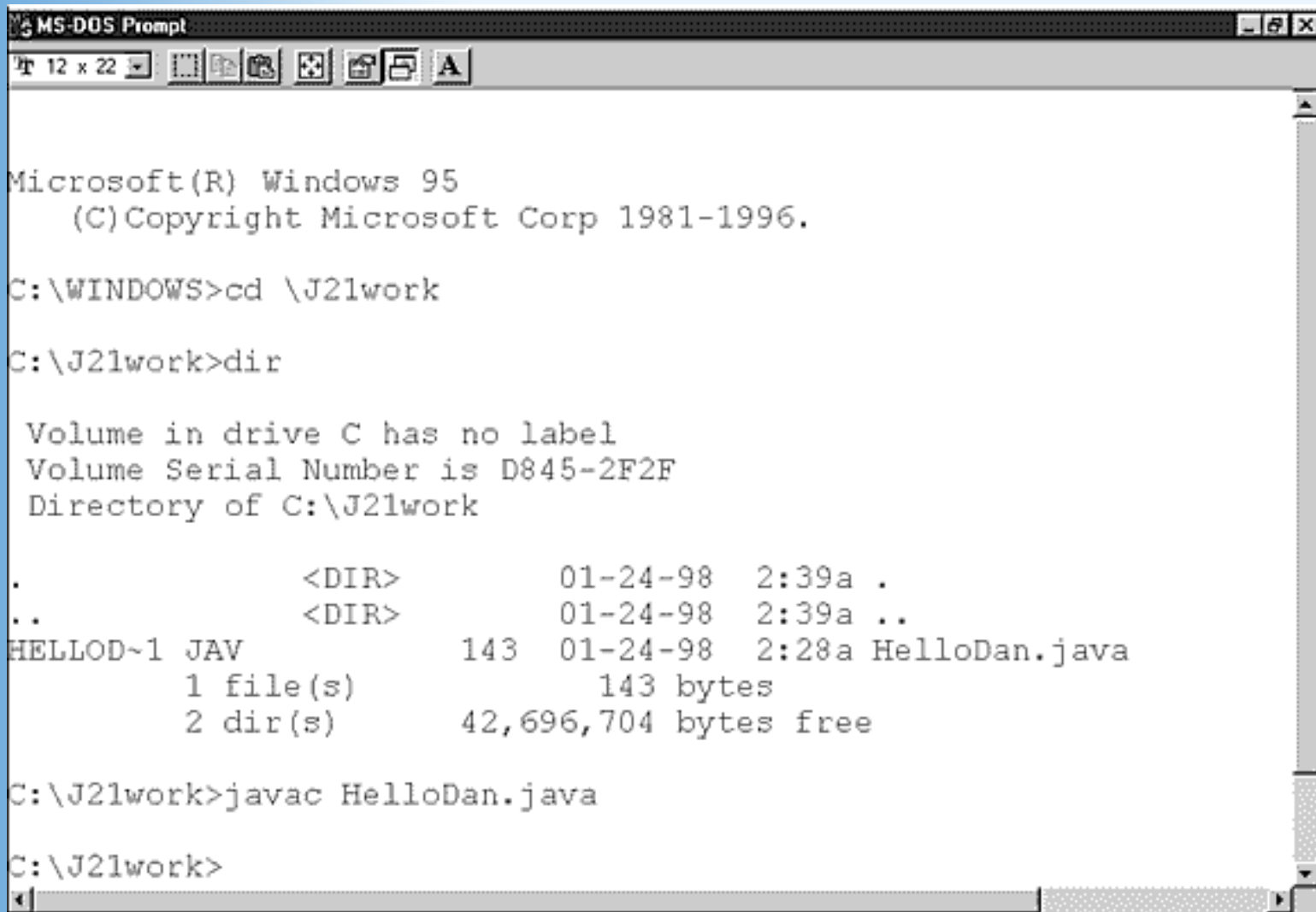
Wenn Sie sich in dem richtigen Ordner befinden, können Sie HelloDan.java kompilieren, indem Sie folgendes an der Eingabeaufforderung eingeben:

```
javac HelloDan.java
```



Wenn Sie das dir-Kommando verwenden, um unter MS-DOS alle Dateien in einem Ordner anzeigen zu lassen, werden Sie mit Sicherheit bemerken, daß die Datei zwei Namen hat - den einen, den Sie ihr gegeben haben, und eine verkürzte Version wie z. B. HELLOD~1.JAV. Diese Abkürzung rührt von der Art her, wie Windows 95 Dateinamen mit mehr als acht Zeichen als Namen und mehr als drei Zeichen für die Erweiterung verwaltet. Wenn Sie mit den JDK-Tools arbeiten, verwenden Sie bei den Kommandos immer den Dateinamen, die Sie einer Datei gegeben haben, und nicht die verkürzte Version.

Abbildung 1.6 zeigt die MS-DOS-Kommandos, mit denen in den Ordner \J21Work gewechselt, die Dateien in dem Ordner aufgelistet und die Datei HelloDan.java kompiliert werden. Auf diesen Weg können Sie sicherstellen, daß Sie die richtigen Kommandos verwenden.



```

MS-DOS Prompt
12 x 22
Microsoft(R) Windows 95
(C) Copyright Microsoft Corp 1981-1996.

C:\WINDOWS>cd \J21work

C:\J21work>dir

Volume in drive C has no label
Volume Serial Number is D845-2F2F
Directory of C:\J21work

.                <DIR>          01-24-98   2:39a .
..              <DIR>          01-24-98   2:39a ..
HELLOD~1  JAVA          143   01-24-98   2:28a HelloDan.java
          1 file(s)              143 bytes
          2 dir(s)          42,696,704 bytes free

C:\J21work>javac HelloDan.java

C:\J21work>

```

Abbildung 1.6:
Java-Programme im MS-DOS-Fenster kompilieren

Der Compiler des JDK gibt keinerlei Meldung aus, wenn das Programm erfolgreich kompiliert wurde. Wenn das Programm ohne Fehler kompiliert werden konnte, dann befindet sich in dem Ordner, der auch HelloDan.java beinhaltet, eine Datei mit dem Namen HelloDan.class. Diese .class-Datei stellt den Bytecode dar, der von der Java Virtual Machine ausgeführt werden kann. Wenn Sie irgendwelche Fehlermeldungen erhalten, sollten Sie noch einmal zu der Quelldatei zurückkehren und prüfen, ob Sie wirklich alles so abgetippt haben, wie es in Listing 1.1 steht.

Sobald Sie eine .class-Datei haben, können Sie diese mit dem Bytecode-Interpreter ausführen. Die JDK-Version des Interpreters hat den Namen java und wird auch von der Eingabeaufforderung aus aufgerufen. Starten Sie HelloDan, indem Sie in den Ordner wechseln, der HelloDan.class beinhaltet, und geben Sie anschließend folgendes Kommando ein:

```
java HelloDan
```



Wenn die Meldung »Class Not Found« angezeigt wird und Sie sich in dem Ordner befinden, in dem sich auch HelloDan.class befindet, dann müssen Sie eventuell eine Einstellung in Ihrer autoexec.bat verändern (siehe Anhang D).

Abbildung 1.7 zeigt die Ausgabe, wenn das Programm erfolgreich ausgeführt werden konnte, und die

Kommandos, um zu diesem Punkt zu gelangen.

```

MS-DOS Prompt
12 x 22
Microsoft(R) Windows 95
(C) Copyright Microsoft Corp 1981-1996.

C:\WINDOWS>cd \J21work

C:\J21work>dir

Volume in drive C has no label
Volume Serial Number is D845-2F2F
Directory of C:\J21work

.                <DIR>          01-24-98   2:39a .
..              <DIR>          01-24-98   2:39a ..
HELLOD~1 JAV      143    01-24-98   2:28a HelloDan.java
HELLOD~1 CLA      486    01-24-98   2:49a HelloDan.class
      2 file(s)                629 bytes
      2 dir(s)             39,419,904 bytes free

C:\J21work>java HelloDan
What's the frequency, Kenneth?

C:\J21work>

```

Abbildung 1.7:
Eine Java-Applikation im MS-DOS-Fenster

Kompilierung und Ausführung unter Solaris

Um die Java-Quelldatei auf einem Solaris-System auszuführen, verwenden Sie den Kommandozeilen-Compiler, der mit dem JDK geliefert wird. Von der Unix-Kommandozeile aus wechseln Sie mit `cd` zu dem Verzeichnis, das die Quelldatei `HelloDan.java` beinhaltet. Wenn Sie das Verzeichnis `J21Work`, das den Windows-Anwendern empfohlen wurde, verwenden, dann benutzen Sie das folgende Kommando:

```
cd ~/J21Work
```

Nachdem Sie sich in dem richtigen Verzeichnis befinden, verwenden Sie das `javac`-Kommando zusammen mit dem Namen der Datei:

```
javac HelloDan.java
```

Solange es keine Fehler gibt, erhalten Sie eine Datei mit dem Namen `HelloDan.class`. Dabei handelt es sich um die Datei mit dem Java-Bytecode, der von der Virtual Machine ausgeführt werden kann. Wenn Sie irgendwelche Fehlermeldungen erhalten, dann sollten Sie noch einmal zu der Quelldatei zurückkehren und prüfen, ob Sie wirklich alles so abgetippt haben, wie es in Listing 1.1 steht.

Sobald Sie eine `.class`-Datei haben, können Sie diese Datei mit dem Bytecode-Interpreter ausführen.

Die JDK-Version der Java-Interpreters heißt `java` und wird ebenfalls von der Kommandozeile aus aufgerufen. Führen Sie `HelloDan` aus, indem Sie in das Verzeichnis wechseln, in dem sich die Datei `HelloDan.class` befindet und folgendes eingeben:

```
java HelloDan
```

Wenn Ihr Programm richtig getippt war und fehlerlos kompiliert wurde, dann sollten Sie den Ausdruck `What's the frequency, Kenneth?` auf dem Bildschirm angezeigt bekommen.



Wenn die Meldung »Class Not Found« angezeigt wird und Sie sich in dem Ordner befinden, in dem sich auch `HelloDan.class` befindet, dann müssen Sie eventuell die Einstellung verändern, wie Ihr System versucht, das JDK zu finden (siehe Anhang D).

Zusammenfassung

Nun da Sie ein Java-Entwicklungstool installiert und es bereits für Ihr erstes Java-Programm verwendet haben, dürfen Sie den Titel »Java-Programmierer« tragen.

Das ist keine Unwahrheit, nach all dem, was Sie am heutigen Tag getan haben. Sie haben nicht nur eine funktionierende Java-Applikation erstellt, sondern auch eine Tour durch die Geschichte von Java mitgemacht und die Stärken, Schwächen und die Zukunft der Sprache kennengelernt.

Java ist eine objektorientierte Programmiersprache, die viel Ähnlichkeit mit C++ besitzt. Sie wurde so entworfen, daß sie einfacher, weniger fehlerträchtig und leichter zu erlernen ist als C++. Sie ist plattformunabhängig und klein, zwei Features, die sie ideal für die Ausführung in World-Wide-Webseiten machen.

Applets sind Programme, die im Web laufen. Applikationen sind hingegen alle anderen Arten von Software, die mit Java geschrieben werden können.

Das ist eine ganze Menge zum Verdauen. Sie sollten jetzt aber die Grundlagen besitzen, um komplexere Applikationen und Ihr erstes Applet zu erstellen. Nach dem morgigen Tag werden Sie auch den Titel »Objektorientierter Programmierer« tragen dürfen.

Fragen und Antworten

Frage:

Welche Beziehung besteht zwischen JavaScript und Java?

Antwort:

Beide haben die ersten vier Buchstaben des Namens gemeinsam. Ein weitverbreitetes Mißverständnis im Web ist, daß JavaScript und Java mehr gemeinsam haben, als das wirklich der Fall ist. Java ist die vielseitige Programmiersprache, die Sie in diesem Buch lernen; Sie verwenden sie, um Applets zu erstellen. JavaScript ist eine von Netscape entwickelte Script-Sprache, die gewisse Ähnlichkeiten mit Java aufweist; mit dieser Sprache können Sie einige schmucke Sachen auf Webseiten veranstalten. Es handelt sich um völlig unabhängige Sprachen, die für unterschiedliche Zwecke eingesetzt werden.

Frage:

Wo kann ich mehr über Java lernen, und wo finde ich Applets und Applikationen, um ein bißchen herumzuspielen?

Antwort:

Sie können den Rest dieses Buches lesen! Im Anschluß an diesen Absatz finden Sie noch einige andere Quellen für Informationen zu Java und Java-Applets:

- Die Java-Homepage (<http://www.java.sun.com>) ist die offizielle Quelle für Informationen zu Java, dem JDK, neuen Releases und zu Entwicklungstools wie dem Java Workshop. Außerdem finden Sie dort umfangreiche Dokumentationen.
- Gamelan (<http://www.gamelan.com>) ist ein Archiv für Applets und Informationen zu Java - alles organisiert in Kategorien. Wenn Sie mit Applets oder Applikationen spielen wollen, dann sollten Sie hier suchen.
- Wenn Sie Diskussionen über Java suchen, dann sollten Sie in der Newsgroup comp.lang.java und den untergeordneten Newsgroups [comp.lang.java.programmer](#) , [comp.lang.java.tech](#), [comp.lang.java.advocacy](#) usw. fündig werden. (Sie benötigen einen UseNet Newsreader, um auf diese Newsgroups zugreifen zu können.)

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).

Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 1

Tag 2

Objektorientierte Programmierung - ein erster Eindruck

Objektorientierte Programmierung ist fast wie Bier.

Die meisten, die das erste Mal ein Glas des malzigen Getränks zu sich nehmen, mögen es nicht und stellen unter Umständen die Zurechnungsfähigkeit derjenigen in Frage, die ein Loblied auf dieses Getränk singen. »Was hab' ich Dir angetan«, fragen sie, »daß Du mich dieses Höllenzeug trinken läßt?«

Nach einiger Zeit kann es durchaus sein, daß die, die trotzdem weiter Bier trinken, es schätzen lernen. (Bei manchen wird dieser Zeitraum Studium genannt.)

Objektorientierte Programmierung ist, wie Bier, ein Geschmack, an den man sich gewöhnen muß. Es ist zum einen eine der bemerkenswertesten Ideen der Programmierung, die in den letzten Jahren eingeführt wurden, und zum anderen der Quell großer Bestürzung bei den Programmierern, die damit nicht vertraut sind.

In gewisser Weise ist dieser Ruf gerechtfertigt. Objektorientierte Programmierung, auch OOP genannt, ist ein Fach, das man jahrelang studieren und üben kann. Die Grundidee ist allerdings einfach: Organisieren Sie Ihre Programme auf eine Art, die die Art widerspiegelt, mit der Objekte in der realen Welt organisiert sind.

Heute werden Sie einen ersten Eindruck davon erhalten, wie Java die Prinzipien der objektorientierten Programmierung verinnerlicht. Die folgenden Themen werden wir behandeln:

- Programme in Form von sogenannten Klassen organisieren und wie diese Klassen verwendet werden, um Objekte zu erzeugen.
- Eine Klasse über zwei Aspekte ihrer Struktur entwerfen: wie sie sich verhalten und über welche Attribute sie verfügen soll.
- Klassen so miteinander verbinden, daß eine Klasse die Funktionalität von einer anderen erbt.
- Klassen über Pakete und Schnittstellen miteinander verbinden.

Wenn Sie mit der objektorientierten Programmierung bereits vertraut sind, wird vieles der heutigen Lektion eine Wiederholung für Sie sein. Selbst, wenn Sie die einführenden Abschnitte überspringen wollen, sollten Sie das Beispielprogramm erstellen, um Erfahrung bei der Erzeugung Ihres ersten Applets zu sammeln.

In Objekten denken

Objektorientierte Programmierung ist in ihrem Kern eine Methode, Computerprogramme zu strukturieren. Sie stellen sich ein Computerprogramm eventuell als Liste von Anweisungen vor, die dem Computer mitteilen, was er zu tun hat, oder als eine Menge kleiner Programme, die auf bestimmte Ereignisse, die der Benutzer auslöst, reagieren.

Die OOP sieht ein Programm auf eine völlig andere Art. Hier ist ein Programm eine Reihe von Objekten, die in vordefinierter Art und Weise zusammenarbeiten, um bestimmte Aufgaben zu erledigen. Nehmen wir LEGO-Steine als Beispiel zur Verdeutlichung.

LEGO-Steine sind - für die unter Ihnen, die keine Kinder haben oder kein inneres Kind, das beschäftigt werden muß - kleine Plastikblöcke, die in unterschiedlichsten Farben und Größen verkauft werden. Diese Steine haben kleine, runde Noppen auf der einen Seite, die fest in die entsprechenden Löcher anderer Steine passen. Über Kombinationen dieser Steine lassen sich größere Formen erzeugen. Es gibt viele verschiedene LEGO-Teile, wie z.B. Räder, Motoren, Gelenke und Flaschenzüge, die man dazu verwenden kann.

Mit LEGO-Bausteinen können Sie alle möglichen Dinge bauen: Burgen, Autos, lange Anhänger, Hosenträger, Sportkleidung... einfach alles, was Sie sich vorstellen können. Jedes LEGO-Steinchen ist ein Objekt, das mit anderen Objekten auf eine ganz bestimmte Art zusammenpaßt, um ein größeres Objekt zu erzeugen.

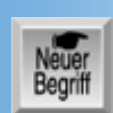
Nehmen wir ein anderes Beispiel. Mit ein bißchen Erfahrung und Hilfe können Sie in den nächsten Computerladen gehen und sich einen kompletten PC aus verschiedenen Einzelteilen zusammenbauen: Motherboard, CPU, Grafikkarte, Festplatte, Tastatur, usw. Idealerweise erhalten Sie, nachdem Sie die einzelnen Komponenten zusammengesetzt haben, ein System, in dem alle Einheiten zusammenarbeiten, um ein größeres System zu bilden. Sie können dieses größere System dann vorrangig zur Lösung der Probleme verwenden, für die Sie den Computer gekauft haben.

Intern kann jede dieser Komponenten ziemlich komplex sein. Auch können sie von verschiedenen Firmen mit unterschiedlichen Methoden entwickelt worden sein. Allerdings müssen Sie nicht wissen, wie die einzelnen Komponenten funktionieren, was jeder einzelne Chip auf der Platine tut oder wie ein »A« an Ihren Computer geschickt wird, wenn Sie auf die (A)-Taste drücken. Jede Komponente, die Sie verwenden, ist eine abgeschlossene Einheit, und als derjenige, der das Gesamtsystem zusammenbaut, sind Sie nur daran interessiert, wie die einzelnen Einheiten miteinander interagieren:

- Wird diese Grafikkarte in einen Slot auf dem Motherboard passen?
- Wird dieser Monitor mit dieser Grafikkarte zusammenarbeiten?
- Werden die einzelnen Komponenten die richtigen Kommandos an die Komponenten senden, mit denen sie zusammenarbeiten, so daß sich die einzelnen Teile des Computers miteinander verstehen?

Sobald Sie die Interaktionen zwischen den einzelnen Komponenten kennen und die entsprechenden Voraussetzungen dafür schaffen, ist es einfach, das Gesamtsystem zusammenzusetzen.

Objektorientierte Programmierung hat sehr viel Ähnlichkeit mit dem Aufbau von Strukturen mit LEGO-Steinen oder dem Zusammenbau eines PC. Bei der OOP bauen Sie Ihre Gesamtprogramme aus unterschiedlichen Komponenten auf, die Objekte genannt werden.

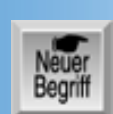


Ein *Objekt* ist ein abgeschlossenes Element eines Computerprogramms, das eine Gruppe miteinander verwandter Features darstellt und dafür ausgelegt ist, bestimmte Aufgaben zu erfüllen. Objekte werden auch als *Instanzen* bezeichnet.

Jedes Objekt hat eine spezielle Rolle in einem Programm, und alle Objekte können auf definierte Arten in einem Programm zusammenarbeiten.

Objekte und Klassen

Die objektorientierte Programmierung wird nach der Beobachtung modelliert, daß in der realen Welt Objekte aus vielen Arten kleinerer Objekte aufgebaut sind. Die Fähigkeit, Objekte zu kombinieren, ist allerdings nur ein allgemeiner Aspekt der objektorientierten Programmierung. Sie umfaßt außerdem Konzepte und Features, die die Erzeugung und den Umgang mit Objekten einfacher und flexibler machen. Das wichtigste dieser Features ist die Klasse.



Eine *Klasse* ist eine Vorlage, die zur Erzeugung vieler Objekte mit ähnlichen Eigenschaften verwendet wird.

Klassen umfassen alle Features eines bestimmten Satzes von Objekten. Wenn Sie ein Programm in einer objektorientierten Sprache schreiben, dann definieren Sie nicht einzelne Objekte, sondern Sie definieren Klassen von Objekten.

Nehmen Sie z.B. die Klasse Tree (engl. Baum), die alle Features aller Bäume beschreibt:

- Hat Blätter und Wurzeln
- Wächst
- Erzeugt Chlorophyll

Die Klasse Tree dient als abstraktes Modell für das Konzept »Baum«. Um ein tatsächliches Objekt in einem Programm zur Verfügung zu haben, das man manipulieren kann, benötigt man konkrete Instanzen der Tree-Klasse.

Klassen werden dazu verwendet, Objekte zu erstellen. Mit diesen Objekten arbeiten Sie dann direkt in einem Programm. Die Klasse Tree kann dazu verwendet werden, eine Vielzahl unterschiedlicher Tree-Objekte zu schaffen, die alle

unterschiedliche Features haben:

- Klein oder groß
- Sehr dichtes Astwerk oder nur sehr spärliches
- Mit Früchten oder ohne

Obwohl diese Objekte sich alle voneinander unterscheiden, haben sie dennoch genug gemein, daß man die Verwandtschaft zwischen ihnen sofort erkennt. Abbildung 2.1 zeigt die Tree-Klasse und einige Objekte, die von dieser Vorlage erzeugt wurden.

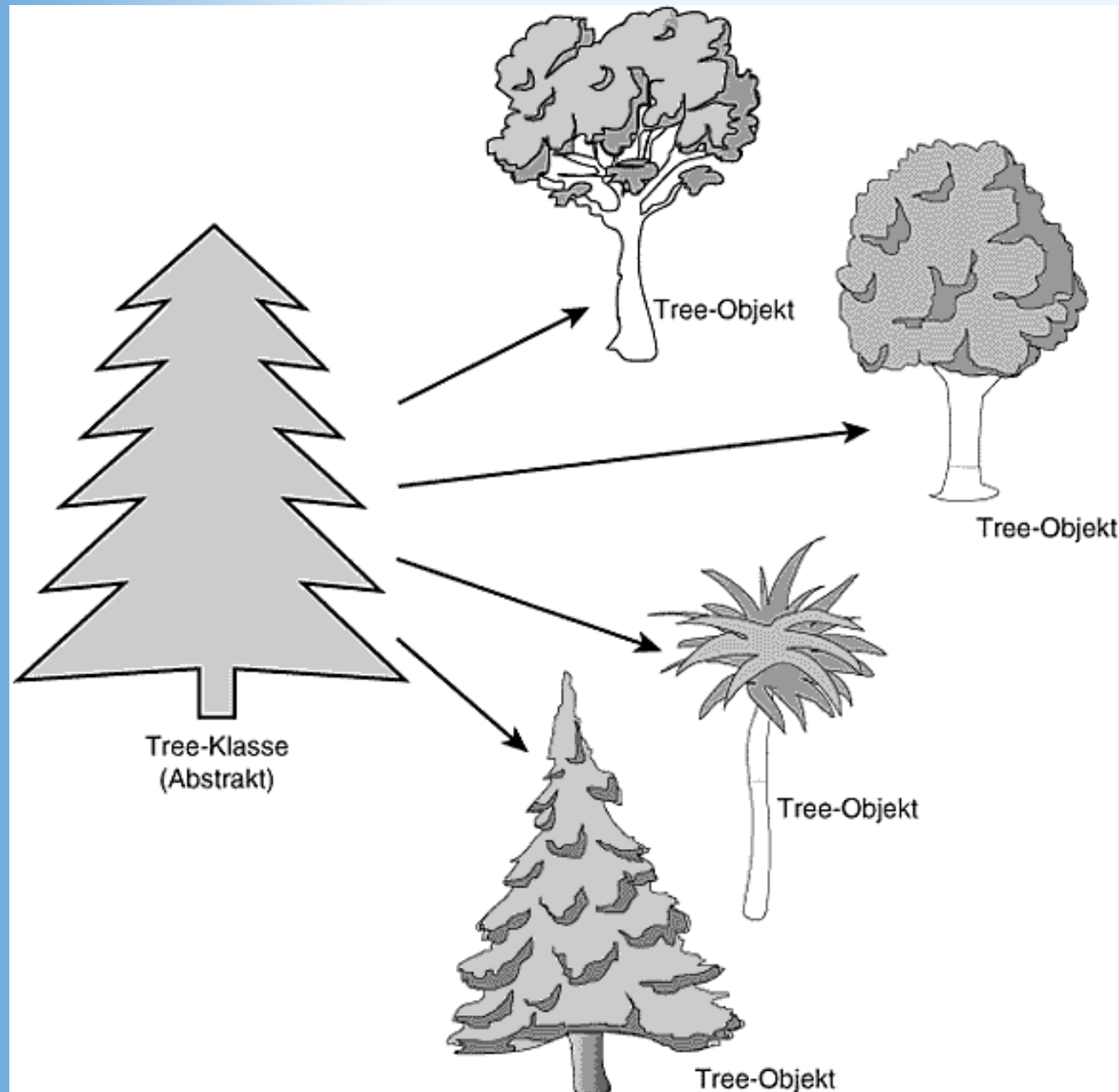


Abbildung 2.1:
Die Tree-Klasse und einige Tree-Objekte

Ein Beispiel für den Entwurf einer Klasse

In einem Beispiel, das eher dem entspricht, was Sie wahrscheinlich mit Java machen werden, könnten Sie eine Klasse für eine Schaltfläche erzeugen. Dies ist ein Element, das in Fenstern, Dialogen und anderen interaktiven Programmen verwendet wird. Die folgenden Eigenschaften könnte die Klasse `CommandButton` definieren:

- Den Text, der den Zweck der Schaltfläche kennzeichnet.
- Die Größe der Schaltfläche.
- Eigenschaften der Erscheinung, wie z.B. ob die Schaltfläche über einen 3D-Schatten verfügt oder nicht.

Die Klasse `CommandButton` könnte zusätzlich noch definieren, wie sich eine Schaltfläche verhalten soll:

- Ob die Schaltfläche einfach oder doppelt angeklickt werden muß, um eine Aktion auszulösen.
- Ob sie Mausklicks eventuell komplett ignorieren soll.
- Was sie tun soll, wenn sie erfolgreich angeklickt wurde.

Sobald Sie die Klasse `CommandButton` definiert haben, können Sie Instanzen der Schaltfläche erzeugen - mit anderen Worten `CommandButton`-Objekte. Die Objekte besitzen alle die Features einer Schaltfläche, wie es in der Klasse definiert ist. Allerdings kann jede Schaltfläche eine andere Erscheinung haben abhängig davon, was für eine Schaltfläche konkret realisiert werden soll.



Eine der Standardklassen von Java - `java.awt.Button` - beinhaltet die gesamte Funktionalität des hypothetischen `CommandButton`-Beispiels und noch mehr. Sie werden am Tag 11 Gelegenheit bekommen, mit dieser Klasse zu arbeiten.

Wenn Sie ein Java-Programm schreiben, dann entwerfen und erstellen Sie eine Reihe von Klassen. Wenn Ihr Programm ausgeführt wird, werden Objekte dieser Klassen erzeugt und nach Bedarf verwendet. Ihre Aufgabe als Java-Programmierer ist es, die richtigen Klassen zu entwerfen, um das umzusetzen, was Ihr Programm tun soll.

Glücklicherweise müssen Sie nicht bei Null beginnen. Jede Version von Java umfaßt eine Gruppe von Klassen, die einen Großteil der Basisfunktionalität, die Sie benötigen, implementieren. Solche Gruppierungen werden als Bibliotheken bezeichnet.



Eine *Klassenbibliothek* ist eine Gruppe von Klassen, die zur Verwendung mit anderen Programmen entworfen wurden. Die Standard-Java-Klassenbibliothek beinhaltet Dutzende von Klassen.

Wenn Sie über die Anwendung der Sprache Java sprechen, dann sprechen Sie eigentlich über die Verwendung der Java-Klassenbibliothek und einiger Schlüsselwörter und Operatoren, die vom Java-Compiler erkannt werden.

Javas Standardbibliothek hat eine große Zahl von Aufgaben, wie z.B. mathematische Funktionen, Umgang mit Text, Grafik, Sound, Interaktion mit dem Benutzer und den Zugriff auf Netzwerke. In vielen Fällen wird die Java-Klassenbibliothek für Ihre Anforderungen ausreichend sein. Hier ist es dann Ihre Aufgabe, eine einzige Klasse zu erstellen, die die Objekte der Standardklassen erzeugt und deren Interaktionen koordiniert.

Für komplexe Java-Programme müssen Sie eventuell eine ganze Reihe neuer Klassen mit definierten Interaktionsmöglichkeiten erzeugen. Diese können Sie dazu verwenden, Ihre eigene Klassenbibliothek zu erstellen, die Sie später auch in anderen Programmen verwenden können.

Die Wiederverwendung ist einer der fundamentalen Vorzüge der objektorientierten Programmierung.

Attribute und Verhaltensweisen

Im allgemeinen besteht jede Klasse, die Sie schreiben, aus zwei Komponenten: Attributen und Verhaltensweisen. In diesem Abschnitt lernen Sie die beiden Komponenten kennen, wie sie in einer theoretischen Klasse mit dem Namen *Jabberwock* angewendet werden. Um diesen Abschnitt zu vervollständigen, werden Sie eine Java-Klasse erstellen, die einen *Jabberwock* - ein drachenartiges Monster aus dem Gedicht *Jabberwocky* von *Lewis Carroll* - implementiert.

Attribute einer Klasse

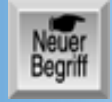
Attribute sind die einzelnen Dinge, die die einzelnen Klassen voneinander unterscheiden. Sie legen auch die Erscheinung, den Zustand und andere Qualitäten der Klasse fest. Überlegen wir uns einmal, wie die theoretische Klasse *Jabberwock* erstellt werden könnte. Ein *Jabberwock* könnte unter anderem die folgenden Attribute aufweisen:

- `color` (Farbe): Orange, Dunkelbraun, Zitronengelb, Dunkelgelb
- `sex` (Geschlecht): männlich, weiblich

- hungry (hungrig): ja, nein

Die Attribute einer Klasse können auch Informationen über den Zustand eines Objektes umfassen. Sie könnten z.B. ein Attribut für den Gemütszustand (wütend oder ruhig) des Jabberwock, den Gesundheitszustand (lebendig oder tot) und das Wahlverhalten (konservativ, liberal oder alternativ) festlegen.

In einer Klasse werden Attribute über Variablen definiert. Sie können sich diese analog zu globalen Variablen für jedes Objekt einer Klasse vorstellen. Jedes Objekt kann andere Werte in seinen Variablen speichern, weshalb diese Variablen auch *Instanzvariablen* genannt werden.



Eine *Instanzvariable* ist ein Stück Information, das ein Attribut eines Objekts definiert. Die Klasse des Objekts definiert die Art des Attributs, und jede Instanz speichert ihren eigenen Wert für dieses Attribut. Instanzvariablen werden auch als *Objektvariablen* bezeichnet.

Jedes Attribut einer Klasse besitzt eine dazugehörige Variable. Sie ändern also dieses Attribut in einem Objekt, indem Sie den Wert dieser Variablen ändern. In dem Programm, das Sie etwas weiter unten erstellen werden, wird die folgende Anweisung verwendet, um anzuzeigen, daß ein Jabberwock-Objekt nicht mehr hungrig ist:

```
j.hungry = false;
```

Instanzvariablen kann bei der Erzeugung eines Objekts ein Wert zugewiesen werden, der während der Lebenszeit des Objekts konstant bleibt. Auf der anderen Seite können Instanzvariablen auch verschiedene Werte zugewiesen werden, während das Objekt in einem laufenden Programm verwendet wird.

Ein anderer Typ von Attributen wird verwendet, um eine ganze Klasse von Objekten anstelle einzelner Objekte dieser Klasse zu beschreiben. Diese werden *Klassenvariablen* genannt.



Eine *Klassenvariable* ist ein Stück Information, das ein Attribut einer Klasse definiert. Die Variable bezieht sich auf die Klasse selbst und all ihre Instanzen, so daß nur ein Wert gespeichert wird unabhängig davon, wie viele Objekte dieser Klasse erzeugt wurden.

Ein gutes Beispiel für eine Klassenvariable ist eine Variable, die zum Zählen der einzelnen Jabberwock-Objekte, die in einem Programm erzeugt wurden, verwendet wird. Wenn für diese Aufgabe eine Instanzvariable in der Jabberwock-Klasse verwendet werden würde, könnte jedes Objekt einen anderen Zählerstand aufweisen. Damit aber nur ein Wert gespeichert werden muß, wird eine Klassenvariable verwendet, auf die jedes Jabberwock-Objekt Zugriff haben kann.

Verhaltensweisen einer Klasse

Das Verhalten ist die Art, mit der eine Klasse bestimmte Dinge gegenüber sich selbst oder anderen Objekten ausführt. Das Verhalten einer Klasse legt fest, was die Objekte dieser Klasse tun, um deren Attribute zu verändern oder wenn andere Objekte sie bitten, etwas zu tun. Ein Jabberwock-Objekt könnte die folgenden Verhaltensweisen beinhalten:

- Ärgerlich werden
- Sich beruhigen
- Einen Bauern fressen
- Das Abendessen ausfallen lassen
- Gesund werden

Das Verhalten einer Klasse wird durch Methoden definiert.



Methoden sind Gruppen von miteinander in Beziehung stehenden Anweisungen in einer Klasse. Diese Anweisungen beziehen sich auf die eigene Klasse und auf andere Klassen und Objekte. Sie werden verwendet, um bestimmte Aufgaben zu erledigen, wie das in anderen Programmiersprachen Funktionen tun.

Objekte kommunizieren miteinander über Methoden. Eine Klasse oder ein Objekt kann Methoden in anderen Klassen oder Objekten aus vielen unterschiedlichen Gründen aufrufen, darunter die folgenden:

- Um ein anderes Objekt von einer Änderung zu berichten.
- Um ein anderes Objekt anzuweisen, etwas an sich selbst zu ändern.
- Um ein anderes Objekt zu bitten, etwas zu tun.

Denken Sie z.B. an den Schwertkämpfer in dem Gedicht »Jabberwocky«. Der folgende Auszug aus dem Gedicht von Lewis Carroll beschreibt genau, was passiert, als der Schwertkämpfer den Jabberwock mit seinem Schwert angreift:

»One, two! One, two! And through and through

The vorpal blade went snicker-snack!

He left it dead, and with its head

He went galumphing back.«

Im Deutschen heißt das soviel wie, der Ritter schlägt mit seinem Schwert dem Jabberwock den Kopf ab und kehrt dann mit dem Kopf im Gepäck dorthin zurück, wo er hergekommen ist.

In Java könnte der Schwertkämpfer als Knight-Objekt (Knight engl. für Ritter) erzeugt werden, mit der Knight-Klasse als Vorlage dafür, wie ein Knight-Objekt sein sollte. Wenn der Ritter dem Jabberwock den Kopf abschlägt, hat dies eine Änderung des inneren Zustandes des Jabberwock zur Folge. Um dieser Änderung Rechnung zu tragen, würde das Knight-Objekt eine Methode verwenden, um dem Jabberwock-Objekt mitzuteilen »Hey! Ich habe Dir Deinen Kopf abgeschlagen. Du bist tot.«

So wie zwischen Instanz- und Klassenvariablen unterschieden wird, gibt es auch Instanz- und Klassenmethoden. Instanzmethoden, die so häufig verwendet werden, daß sie einfach nur Methoden genannt werden, gehören zu einem Objekt einer Klasse. Wenn eine Methode ein einzelnes Objekt verändert, dann muß diese eine Instanzmethode sein. Klassenmethoden gehören zu einer Klasse selbst.

Eine Klasse erstellen

Nachdem die grundlegende Terminologie der objektorientierten Programmierung nun eingeführt wurde, wird alles vielleicht anhand eines konkreteren Beispiels klarer. Sie werden ein funktionierendes Beispiel der Jabberwock-Klasse erstellen, damit Sie sehen, wie Instanzvariablen und Methoden in einer Klasse definiert werden. Sie werden auch ein Java-Applet erstellen, das ein neues Objekt der Jabberwock-Klasse erzeugt, die Werte der Instanzvariablen dieses Objektes verändert und bestimmte Aktionen aufgrund der Werte der Instanzvariablen ausführt.



Auf die eigentliche Syntax dieses Beispiels wird hier nicht sehr ausführlich eingegangen. Sehen Sie dies hier mehr als Einführung in die objektorientierte Programmierung und weniger als eine Lektion über die Syntax von Java, in die Sie am dritten Tag eintauchen werden.

Öffnen Sie den Texteditor, den Sie zur Erstellung von Java-Programmen verwenden, so daß Sie mit der Erzeugung der Quelldatei beginnen können. Anstatt ein ganzes Programm einzugeben, werden Sie einige Anweisungen eingeben, während Sie etwas über deren Verwendung lernen. Sie erhalten am Ende die Gelegenheit, Ihre Arbeit zu überprüfen, um sicherzugehen, daß alles korrekt ist.

Den Beginn stellt eine elementare Klassendefinition dar. Geben Sie folgendes ein:

```
class Jabberwock {  
}
```

Nun haben Sie eine Klasse erzeugt. Momentan tut sie noch nicht viel, aber die beiden Zeilen sind ein Beispiel für die einfachste Klassendefinition in Java.

Um Jabberwock etwas spezieller zu gestalten, erstellen Sie drei Instanzvariablen für diese Klasse. Direkt unterhalb der Zeile `class Jabberwock {` fügen Sie die folgenden drei Zeilen ein:

```
String color;
String sex;
boolean hungry;
```

Diese drei Zeilen erzeugen drei Instanzvariablen. Zwei davon, `color` (Farbe) und `sex` (Geschlecht), können String-Objekte beinhalten. Ein String ist ein allgemeiner Begriff, der für eine Gruppe von Zeichen steht. In Java wird ein String-Objekt mit einer der Standardklassen aus der Java-Klassenbibliothek erzeugt. Die String-Klasse wird zur Speicherung von Text verwendet und bietet diverse Funktionen zur Bearbeitung von Text.

Die dritte Variable, `hungry` (hungrig), ist eine boolesche Variable, die nur zwei verschiedene Werte annehmen kann: `true` (wahr) oder `false` (falsch). Dieses Objekt wird dafür verwendet, anzuzeigen, ob der Jabberwock hungrig ist (`true`) oder voll (`false`).



Boolesche Werte sind spezielle Variablentypen, die nur die Werte `true` oder `false` aufnehmen können. Im Gegensatz zu anderen Sprachen haben boolesche Werte keine numerischen Werte, wobei 1 `true` und 0 `false` entspricht. Der Ausdruck `boolesch` geht auf George Boole, einen irischen Mathematiker zurück, der von 1815 bis 1864 lebte. Ein anderer Begriff, der auf ihn zurückgeht, ist die Boolesche Algebra, die fundamental für die Programmierung von Computern, die digitale Elektronik und die Logik ist.

Sie können der Jabberwock-Klasse Verhaltensweisen hinzufügen, in dem Sie Methoden implementieren. Ein Jabberwock kann alle möglichen Dinge tun (mit den Krallen zupacken, zubeißen und so weiter); um das Ganze kurz zu halten, wollen wir nur zwei Methoden hinzufügen - eine, die das Monster füttert, und eine andere, um die Attribute des Monsters zu überprüfen.

Zum Start fügen Sie die folgenden Zeilen unter den drei Instanzvariablen in Ihrer Klassendefinition ein:

```
void feedJabberwock() {
    if (hungry == true) {
        System.out.println("Yum -- a peasant.");
        hungry = false;
    } else
        System.out.println("No, thanks -- already ate.");
}
// In Kürze mehr
```



Die letzte Zeile `// In Kürze mehr` ist eine Kommentarzeile. Kommentare werden als Information für diejenigen eingefügt, die den Quellcode lesen, um herausfinden, was dieser tut. Computer sind daran überhaupt nicht interessiert - alles beginnend mit `//` bis zum Ende der Zeile wird von einem Java-Compiler ignoriert. In der Jabberwock-Klasse wird der Kommentar als Platzhalter verwendet. Sie werden diesen bald ersetzen.

Die Methode `feedJabberwock()` prüft, ob ein Jabberwock-Objekt hungrig ist (in der Zeile `if (hungry == true)`). Wenn es hungrig ist, wird das Objekt gefüttert (zu seiner großen Freude), und der Status von `hungry` wird auf `false` gesetzt. Wenn das Objekt nicht hungrig ist, wird eine Meldung angezeigt, daß das Monster bereits gegessen hat. Das gesamte Programm sollte bis hierher wie folgt aussehen:

Listing 2.1: Der aktuelle Text von Jabberwock.java

```
1: class Jabberwock {
2:     String color;
3:     String sex;
4:     boolean hungry;
5:
6:     void feedJabberwock() {
```



```

7:         if (hungry == true) {
8:             System.out.println("Yum -- a peasant!");
9:             hungry = false;
10:        } else
11:            System.out.println("No, thanks -- already ate.");
12:    }
13:
14: // In Kürze mehr
15:}

```



Die Einrückungen und Leerzeilen, die im Quelltext für Platz sorgen, werden von einem Java-Compiler nicht beachtet. Wie Kommentare werden auch diese für den Programmierer eingefügt, damit sich die Logik eines Programms leichter nachvollziehen läßt. So wie die Einrückungen und die Abstände hier verwendet wurden (Leerzeilen zwischen Methoden und Einrückungen bei Methoden und Variablen), werden sie im gesamten Buch verwendet. Die Java-Klassenbibliothek verwendet eine ähnliche Einrückung. Sie können einen beliebigen Stil dabei verwenden.

Bevor Sie diese Klasse kompilieren, müssen Sie eine weitere Methode hinzufügen. Die Methode `showAttributes()` zeigt die aktuellen Werte der Instanzvariablen einer Instanz der Klasse `Jabberwock`.

Löschen Sie in dem Programm die Kommentarzeile `// In Kürze mehr`, und ersetzen Sie sie durch den folgenden Code:

```

void showAttributes() {
    System.out.printl ("This is a " + sex + " " + color + " jabberwock.");
if (hungry == true)
    System.out.println("The jabberwock is hungry.");
else
    System.out.println("The jabberwock is full.");
}

```

Die Methode `showAttributes()` gibt zwei Zeilen auf dem Bildschirm aus: in der ersten die Werte der Variablen `sex` und `color` und in der zweiten, ob das `Jabberwock` hungrig ist. Speichern Sie die Quelldatei in Ihrem Texteditor, und achten Sie darauf, daß die Datei auch `Jabberwock.java` heißt, damit der Dateiname mit dem Namen der Klasse übereinstimmt.

An diesem Punkt verfügen Sie über eine `Jabberwock`-Klasse mit Instanzvariablen und Instanzmethoden, die zur Anzeige und Veränderung der Werte dieser Variablen verwendet werden können.

Kompilieren Sie das Programm mit einer der beiden folgenden Methoden - abhängig davon, welches System Sie verwenden.

Windows:

Wechseln Sie von der MS-DOS-Eingabeaufforderung mit dem `CD`-Kommando in den Ordner, in dem sich Ihre Java-Quelldatei befindet. Und verwenden Sie anschließend das Kommando `javac`, um die Datei zu kompilieren:

```
javac Jabberwock.java
```

Solaris:

Wechseln Sie von der Kommandozeile aus mit dem `CD`-Kommando in das Verzeichnis, in dem sich Ihre Java-Quelldatei befindet. Und verwenden anschließend das Kommando `javac`, um die Datei zu kompilieren:

```
javac Jabberwock.java
```

Wenn bei der Kompilierung Probleme auftreten, dann prüfen Sie anhand von Listing 2.2, ob Sie eventuell Tippfehler gemacht haben.

Listing 2.2: Der aktuelle Text von `Jabberwock.java`

```

1: class Jabberwock {
2:     String color;
3:     String sex;
4:     boolean hungry;

```

```

5:
6:     void feedJabberwock() {
7:         if (hungry == true) {
8:             System.out.println("Yum -- a peasant!");
9:             hungry = false;
10:        } else
11:            System.out.println("No, thanks -- already ate.");
12:    }
13:
14:    void showAttributes() {
15:        System.out.println("This is a " + sex + " " + color + " jabberwock.");
16:        if (hungry == true)
17:            System.out.println("The jabberwock is hungry.");
18:        else
19:            System.out.println("The jabberwock is full.");
20:    }
21: }

```

Das Programm ausführen

Wenn Sie die Datei Jabberwock.java mit einem Kommandozeilen-Tool wie dem Java-Interpreter ausführen, erhalten Sie einen Fehler:

```
In class Jabberwock: void main(String argv[]) is not definedp
```

Dieser Fehler tritt auf, da der Java-Interpreter davon ausgeht, daß das Programm eine Applikation ist, wenn Sie es von der Kommandozeile aus aufrufen. Wenn eine Applikation ausgeführt wird, ist deren main()-Methode der Startpunkt des Programms. Da die Klasse Jabberwock keine main()-Methode besitzt, weiß der Interpreter nicht, was er mit ihr machen soll.

Es gibt zwei Möglichkeiten, die Jabberwock-Klasse zu verwenden:

- Erzeugen Sie eine separates Java-Applet oder eine Java-Applikation, die diese Klasse verwendet.
- Fügen Sie eine main()-Methode in die Jabberwock-Klasse ein, so daß diese direkt ausgeführt werden kann.

In dieser Übung wollen wir letzteres tun. Laden Sie Jabberwock.java in den Texteditor, und fügen Sie eine Leerzeile direkt über der letzten Zeile des Programms ein (Zeile 21 in Listing 2.2).

Geben Sie hier nun folgendes ein:

```

public static void main (String arguments[]) {
    Jabberwock j = new Jabberwock();
    j.color = "orange";
    j.sex = "male";
    j.hungry = true;
    System.out.println("Calling showAttributes ...");
    j.showAttributes();
    System.out.println("-----");
    System.out.println("Feeding the jabberwock ...");
    j.feedJabberwock();
    System.out.println("-----");
    System.out.println("Calling showAttributes ...");
    j.showAttributes();
    System.out.println("-----");
    System.out.println("Feeding the jabberwock ...");
    j.feedJabberwock();
}

```

Mit der main()-Methode kann die Jabberwock-Klasse nun als Applikation verwendet werden. Speichern Sie die Datei, und kompilieren Sie sie anschließend.

Das Listing 2.3 zeigt die endgültige Version der Datei Jabberwock.java, für den Fall, daß Sie bei der Kompilierung Probleme haben.



Sie finden auf der CD eine Kopie der Quelldateien und anderer benötigter Dateien. Wenn Sie Probleme mit Programmen in diesem Buch haben, können Sie anhand dieser Dateien prüfen, wo das Problem liegt.

Listing 2.3: Die endgültige Version von Jabberwock.java

```

1: class Jabberwock {
2:     String color;
3:     String sex;
4:     boolean hungry;
5:
6:     void feedJabberwock() {
7:         if (hungry == true) {
8:             System.out.println("Yum -- a peasant!");
9:             hungry = false;
10:        } else
11:            System.out.println("No, thanks -- already ate.");
12:    }
13:
14:    void showAttributes() {
15:        System.out.println("This is a " + sex + " " + color + " jabberwock.");
16:        if (hungry == true)
17:            System.out.println("The jabberwock is hungry.");
18:        else
19:            System.out.println("The jabberwock is full.");
20:    }
21:
22:    public static void main (String arguments[]) {
23:        Jabberwock j = new Jabberwock();
24:        j.color = "orange";
25:        j.sex = "male";
26:        j.hungry = true;
27:        System.out.println("Calling showAttributes ...");
28:        j.showAttributes();
29:        System.out.println("-----");
30:        System.out.println("Feeding the jabberwock ...");
31:        j.feedJabberwock();
32:        System.out.println("-----");
33:        System.out.println("Calling showAttributes ...");
34:        j.showAttributes();
35:        System.out.println("-----");
36:        System.out.println("Feeding the jabberwock ...");
37:        j.feedJabberwock();
38:    }
39: }
```

Anhand von Listing 2.3 wollen wir im folgenden ansehen, was in der main()-Methode passiert:

- Zeile 22: Die main()-Methode wird deklariert. Die erste Zeile der main()-Methode sieht immer so aus. Die einzelnen Elemente dieses Ausdrucks lernen Sie später in dieser Woche kennen.
- Zeile 23: `Jabberwock j = new Jabberwock();`, erzeugt eine neue Instanz der Jabberwock-Klasse und speichert eine Referenz zu dieser in einer neuen Variablen mit dem Namen j. Wie Sie bereits gelernt haben, arbeiten Sie in Java-Programmen normalerweise nicht direkt mit Klassen. Statt dessen erzeugen Sie Objekte dieser Klassen und rufen Methoden dieser Objekte auf, um mit den Objekten zu arbeiten.
- Die Zeilen 24-26: Den Instanzvariablen color, sex und hungry des Jabberwock- Objektes, das in Zeile 23 erzeugt wurde, werden Werte zugewiesen. color erhält den Wert »orange«, sex den Wert »male« und die Variable hungry

den booleschen Wert true. Dies zeigt an, daß dieses Jabberwock-Objekt hungrig ist.

- Zeile 27: In dieser Zeile und einigen weiteren, die folgen, wird die Anweisung `System.out.println()` verwendet, um Informationen auf dem Bildschirm auszugeben. Alles, was sich dabei zwischen den Klammern befindet, wird angezeigt.
- Zeile 28: Die Methode `showAttributes()`, die in dem Jabberwock-Objekt definiert ist, wird hier aufgerufen. Die fordert das Jabberwock-Objekt dazu auf, die Werte seiner Variablen `color`, `sex` und `hungry` auszugeben.
- Zeile 31: Hier wird die Methode `feedJabberwock()` des Jabberwock-Objekts aufgerufen, die den Wert der Variable `hungry` von true auf false setzt und eine Bemerkung voll des Dankes von dem Jabberwock-Objekt anzeigt: »Yum -- a peasant!« (zu deutsch »Lecker -- ein Bauer!«)
- Zeile 33: Die Methode `showAttributes()` wird hier erneut aufgerufen, um die Werte der Instanzvariablen des Jabberwock-Objekts auszugeben. Diesmal sollte die Meldung aussagen, daß das Jabberwock voll ist (`hungry` hat den Wert false).
- Zeile 36: Die Methode `feedJabberwock()` wird erneut aufgerufen, um einen Versuch zu starten, das Jabberwock zu füttern. Da das Jabberwock aber bereits voll ist, lehnt es mit einem höflichen »No thanks -- already ate.« (zu deutsch »Nein danke -- habe schon gegessen.«) die Nahrung ab.

Die Jabberwock-Applikation können Sie mit einer der folgenden plattformspezifischen Prozeduren ausführen:

Windows:

Wechseln Sie von der MS-DOS-Eingabeaufforderung aus mit dem CD-Kommando in das Verzeichnis, das die Datei `Jabberwock.class` enthält, und starten Sie den Interpreter mit dem Befehl `java`:

```
java Jabberwock
```

Solaris:

Wechseln Sie von der Kommandozeile aus mit dem CD-Kommando in das Verzeichnis, das die Datei `Jabberwock.class` enthält, und starten Sie den Interpreter mit dem Befehl `java`:

```
java Jabberwock
```

Wenn Sie die Jabberwock-Klasse ausführen, sollten Sie die folgende Ausgabe auf dem Bildschirm erhalten:

```
Calling showAttributes ...
This is a male orange jabberwock.
The jabberwock is hungry.
-----
Feeding the jabberwock ...
Yum -- a peasant!
-----
Calling showAttributes ...
This is a male orange jabberwock.
The jabberwock is full.
-----
Feeding the jabberwock ...
No, thanks -- already ate.
```



Ich gehe hier davon aus, daß Sie wissen, wie eine Java-Applikation kompiliert und ausgeführt wird. Eine umfangreichere Anleitung finden Sie am Tag 1 und in der Dokumentation Ihres Entwicklungstools.

Klassen und deren Verhalten organisieren

Eine Einführung in die objektorientierte Programmierung mit Java ist nicht komplett ohne eine erste Betrachtung der folgenden drei Konzepte: Vererbung, Schnittstellen und Pakete.

Diese drei Konzepte stellen allesamt Mechanismen zur Organisation von Klassen und dem Verhalten von Klassen dar. Die Klassenbibliothek von Java verwendet diese Konzepte, und auch die Klassen, die sie für Ihre eigenen Programme

erstellen, werden diese benötigen.

Vererbung

Die Vererbung ist eines der entscheidenden Konzepte der objektorientierten Programmierung und beeinflusst direkt die Art und Weise, wie Sie Ihre eigenen Java-Klassen schreiben.



Vererbung ist ein Mechanismus, der es einer Klasse ermöglicht, all Ihre Verhaltensweisen und Attribute von einer anderen Klasse zu erben.

Über die Vererbung verfügt eine Klasse sofort über die gesamte Funktionalität einer vorhandenen Klasse. Aus diesem Grund kann man eine neue Klasse erstellen, indem man angibt, wie sie sich von einer bestehenden unterscheidet.

Durch die Vererbung werden alle Klassen Teil einer strengen Hierarchie - Klassen, die Sie erzeugen, Klassen aus der Klassenbibliothek von Java und anderen Bibliotheken.



Eine Klasse, von der andere Klassen abgeleitet werden (sprich, die ihre Funktionalität an andere Klassen vererbt), wird *Superklasse* genannt. Die Klasse, die die Funktionalität erbt, wird als *Subklasse* bezeichnet.

Eine Klasse kann lediglich eine Superklasse besitzen. Jede Klasse kann allerdings eine uneingeschränkte Anzahl von Subklassen haben. Subklassen erben alle Attribute und sämtliche Verhaltensweisen ihrer Superklasse.

In der Praxis bedeutet dies, daß, wenn eine Superklasse Verhaltensweisen und Attribute besitzt, die Ihre Klasse benötigt, Sie diese nicht neu definieren oder den Code kopieren müssen, um über dieselbe Funktionalität in Ihrer Klasse zu verfügen. Ihre Klasse erhält all dies automatisch von ihrer Superklasse, die Superklasse erhält das Ganze wiederum von ihrer Superklasse und so weiter, der Hierarchie folgend. Ihre Klasse wird eine Kombination aller Features der Klassen über ihr in der Hierarchie und ihrer eigenen.

Diese Situation läßt sich sehr gut damit vergleichen, wie Sie bestimmte Eigenschaften, wie z.B. Größe, Haarfarbe, die Leidenschaft für Ska-Musik und den Widerwillen, nach dem Weg zu fragen, von Ihren Eltern geerbt haben. Diese haben einige dieser Dinge wiederum von ihren Eltern geerbt, die von den ihren und so weiter bis zurück in den Garten Eden, zum Urknall oder was auch immer am Anfang war.

Die Abbildung 2.2 zeigt, wie eine Hierarchie von Klassen organisiert ist.

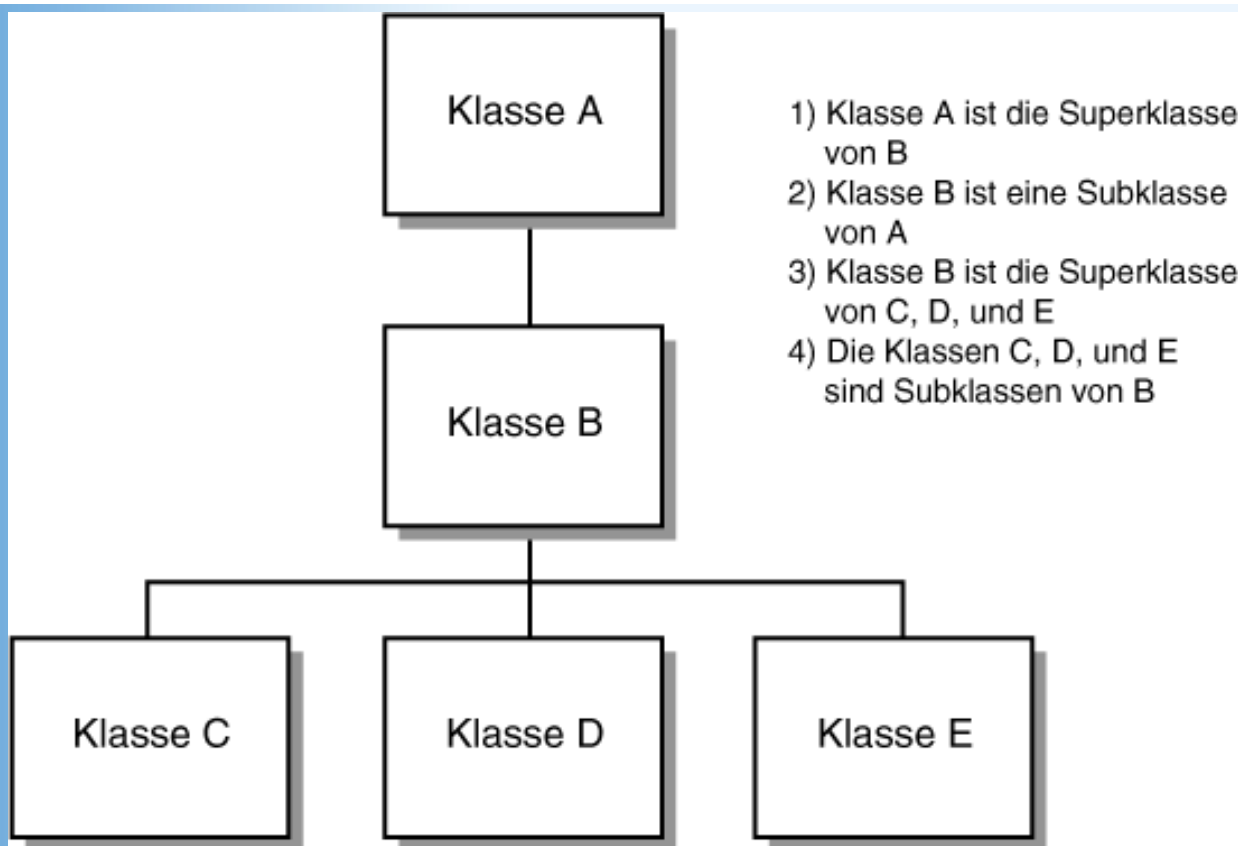


Abbildung 2.2:
Eine Klassenhierarchie

An der Spitze der Hierarchie der Java-Klassen steht die Klasse Object - alle Klassen werden von dieser Superklasse abgeleitet. Object stellt die allgemeinste Klasse in der Hierarchie dar und legt die Verhaltensweisen und Attribute, die an alle Klassen in der Java-Klassenbibliothek vererbt werden, fest. Jede Klasse, die sich weiter unten in der Hierarchie befindet, ist stärker auf einen bestimmten Zweck zugeschnitten. Eine Klassenhierarchie definiert abstrakte Konzepte an der Spitze der Hierarchie. Diese Konzepte werden, je weiter Sie in der Hierarchie hinabsteigen, immer konkreter.

Oft werden Sie, wenn Sie in Java eine neue Klasse erstellen, die gesamte Funktionalität einer existierenden Klasse mit einigen eigenen Modifikationen haben wollen. Es könnte z.B. sein, daß Sie eine Version von `CommandButton`-Schaltflächen wollen, die beim Anklicken ein ohrenbetäubendes Explosionsgräusch erzeugen. (Weder die Autoren noch der Verlag halten dies für eine gute Idee, noch können sie für etwaige Schäden beim Anwender haftbar gemacht werden.)

Um die gesamte Funktionalität von `CommandButton` ohne den Aufwand, sie neu erstellen zu müssen, zu erhalten, können Sie eine Klasse als Subklasse von `CommandButton` definieren. Ihre Klasse verfügt dann automatisch über die Attribute und Verhaltensweisen, die in `CommandButton`, und über die, die in den Superklassen von `CommandButton` definiert wurden. Sie müssen sich jetzt nur noch um das kümmern, was Ihre neue Klasse von `CommandButton` unterscheidet. Der Mechanismus zur Definition neuer Klassen über die Unterschiede zwischen diesen und deren Superklasse wird im Englischen als *Subclassing* bezeichnet.

Sollte Ihre Klasse ein komplett neues Verhalten definieren und keine Subklasse einer bestehenden Klasse sein, können Sie diese direkt von der Klasse Object ableiten. Dies erlaubt es ihr, sich nahtlos in die Klassenhierarchie von Java zu integrieren. Wenn Sie eine Klassendefinition erstellen, die keine Superklasse angibt, nimmt Java an, daß die neue Klasse direkt von Object abgeleitet wird. Die Klasse `Jabberwock`, die Sie weiter oben erstellt haben, ist direkt von Object abgeleitet.

Eine Klassenhierarchie erzeugen

Wenn Sie eine große Menge von Klassen erzeugen, ist es sinnvoll, diese zum einen von den bestehenden Klassen der Klassenhierarchie abzuleiten, und zum anderen, daß diese eine eigene Hierarchie bilden. Ihre Klassen auf diese Art und Weise zu organisieren bedarf einer umfangreichen Planung. Als Entschädigung erhalten Sie allerdings unter anderem die folgenden Vorteile:

- Funktionalitäten, die mehrere Klassen gemein haben, können in einer Superklasse zusammengefaßt werden.

Dadurch wird es möglich, diese Funktionalitäten in allen Klassen, die sich in der Hierarchie unterhalb dieser Superklasse befinden, wiederholt zu verwenden.

- Änderungen in einer Superklasse werden automatisch in all ihren Subklassen, deren Subklassen usw. widergespiegelt. Es ist nicht nötig, die Klassen, die sich in der Hierarchie darunter befinden, zu ändern oder neu zu kompilieren, da diese die neuen Informationen über die Vererbung erhalten.

Stellen Sie sich einmal vor, Sie hätten eine Klasse erstellt, die sämtliche Features eines Jabberwock implementiert. (Dies sollte nicht allzuschwer sein, wenn Sie den entsprechenden Teil des heutigen Tages nicht übersprungen haben.)

Die Klasse Jabberwock ist vollständig, arbeitet erfolgreich, und alles ist fein. Nun wollen Sie eine Klasse erstellen, die Dragon (engl. für Drachen) heißt. Drachen und Jabberwocks besitzen einige ähnliche Features - beides sind sehr große Monster und fressen Bauern. Beide haben Krallen, kräftige Zähne und eine ausgeprägte Persönlichkeit. Ihre erste Idee ist nun vielleicht, die Quelldatei Jabberwock.java zu öffnen und einen großen Teil des Quellcodes in eine neue Quelldatei mit dem Namen Dragon.java zu kopieren.

Eine wesentlich bessere Idee wäre es allerdings, die gemeinsame Funktionalität von Dragon und Jabberwock zu ermitteln und sie in einer etwas allgemeineren Klassenhierarchie zu organisieren. Dies wäre für die Klassen Dragon und Jabberwock eventuell etwas viel Aufwand. Was aber, wenn Sie noch die Klassen Medusa, Yeti, Sasquatch, Grue und DustBunny hinzufügen wollen? Allgemeine Verhaltensweisen und Attribute in eine oder mehrere wiederverwendbare Superklassen einzufügen, reduziert den Gesamtaufwand ganz beträchtlich.

Um eine Klassenhierarchie zu entwerfen, die diese Aufgabe erfüllt, beginnen wir an der Spitze mit der Klasse Object, dem Ausgangspunkt aller Klassen unter Java. Die allgemeinste Klasse, der sowohl ein Jabberwock als auch ein Drachen angehören, könnte Monster genannt werden. Ein Monster könnte allgemein als grausame Kreatur gelten, die Menschen terrorisiert und die Werte von Eigenschaften verringert. In der Monster-Klasse definieren Sie nur das Verhalten, das etwas als grausam, terrorisierend und schlecht für die Nachbarschaft beschreibt.

Unterhalb von Monster könnte es zwei Klassen geben: FlyingMonster (fliegendes Monster) und WalkingMonster (gehendes Monster). Der offensichtlichste Unterschied zwischen diesen beiden Klassen ist, daß Monster der einen Klasse fliegen können und die Monster der anderen Klasse nicht. Die Verhaltensweisen der fliegenden Monster könnten die folgenden einschließen: auf die Beute stürzen, Bauern ergreifen und mit in die Lüfte tragen, diese aus großen Höhen abstürzen lassen usw. Gehende Monster würden sich hingegen ganz anders verhalten. Abbildung 2.3 zeigt, was wir bis jetzt haben.

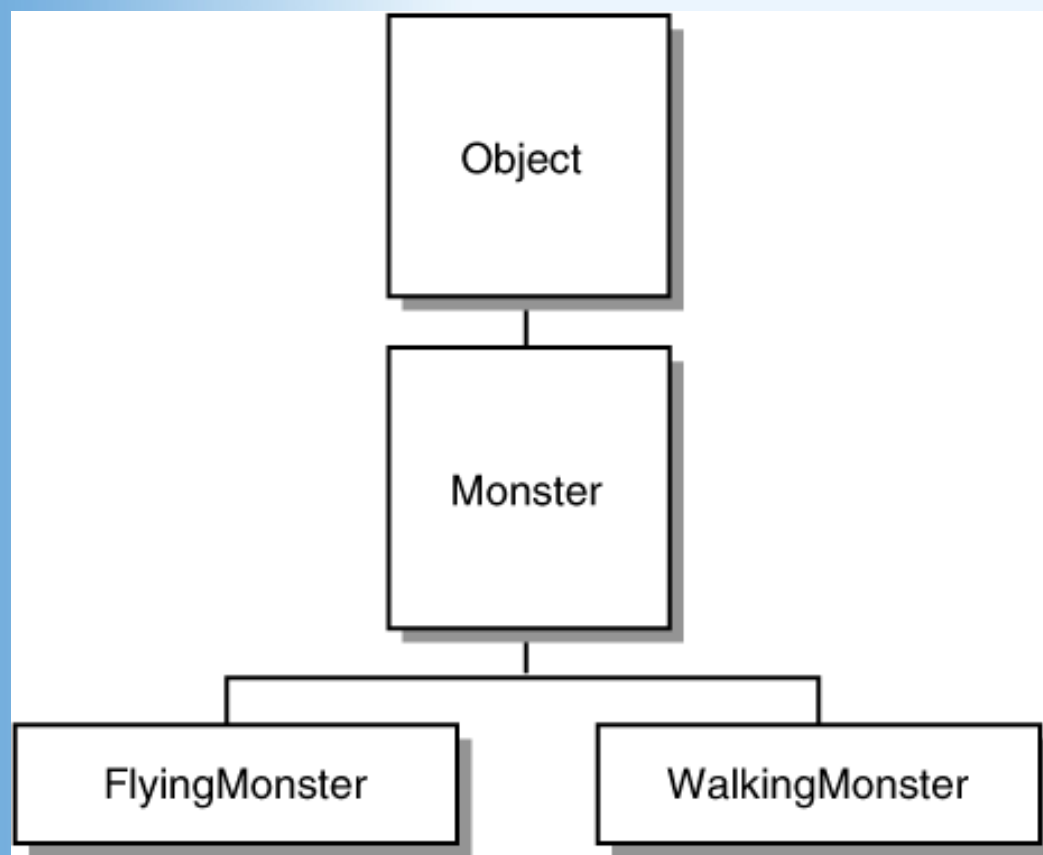


Abbildung 2.3:

Die grundlegende Monster-Hierarchie

Davon ausgehend, kann die Hierarchie noch spezieller werden. Von FlyingMonster könnten Sie einige Klasse ableiten: Mammal, Reptile, Amphibian usw. Alternativ könnten Sie auch weitere Funktionalitäten ausmachen und in den Zwischenklassen TwoLegged und FourLegged für zwei- und vierbeinige Monster mit jeweils unterschiedlichen Verhaltensweisen zusammenfassen (siehe auch Abbildung 2.4).

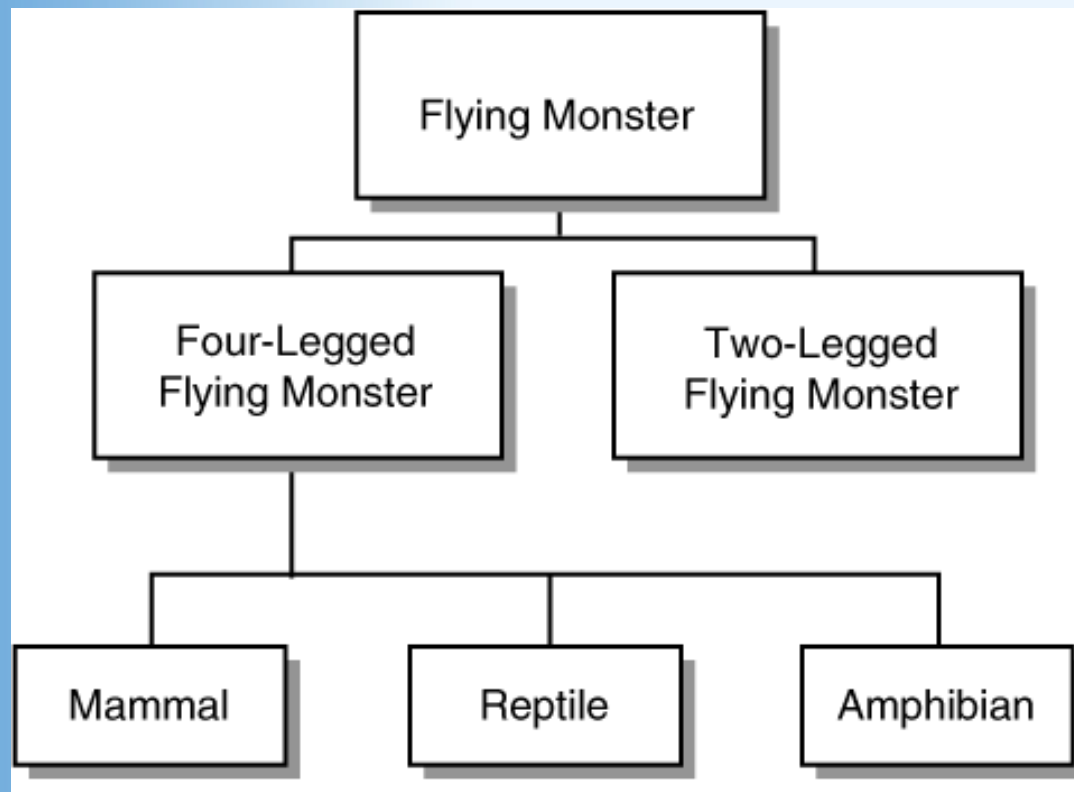


Abbildung 2.4:
Zwei- und vierbeinige fliegende Monster

Zu guter Letzt steht die Hierarchie, und Sie haben einen Platz für Jabberwock. Dies wäre eine Subklasse von Reptile, Fourlegged, FlyingMonster, Monster und Object, da FlyingMonster eine Subklasse von Monster und Monster eine Subklasse von Object ist.

Wo kommen nun Eigenschaften, wie Geschlecht, Farbe oder Appetit ins Spiel? Dort, wo Sie sich in die Klassenhierarchie am harmonischsten einfügen. Sie können sex und color als Instanzvariablen in Monster definieren, und alle Subklassen werden ebenfalls über diese Variablen verfügen. Denken Sie daran, daß Sie eine Verhaltensweise oder ein Attribut nur einmal in der Hierarchie definieren müssen und alle Subklassen es dann automatisch erben.



Der Entwurf einer effektiven Klassenhierarchie schließt eine umfangreiche Planung und Überarbeitung ein. Während Sie versuchen, Attribute und Verhaltensweisen in einer Hierarchie anzuordnen, werden Sie wahrscheinlich oftmals erkennen, daß es sinnvoll ist, Klassen innerhalb der Hierarchie an andere Stellen zu verschieben. Das Ziel ist es, die Anzahl sich wiederholender Features auf das Nötigste zu reduzieren. Wenn Sie eine Hierarchie von Monstern entwickeln, werden Sie eventuell Mammal, Reptile und Amphibian direkt unterhalb von Monster anordnen wollen, falls dies die Funktionalität, die Ihre Klassen beinhalten, besser beschreiben sollte.

Vererbung in Aktion

Die Vererbung funktioniert in Java wesentlich einfacher als in der realen Welt. Hier sind weder Testamentsvollstrecker noch Richter oder Gerichte irgendeiner Art nötig.

Wenn Sie ein neues Objekt erzeugen, ermittelt Java jede Variable, die für dieses Objekt definiert wurde, und jede Variable, die in jeder der Superklassen des Objekts definiert wurde. Auf diesem Weg werden alle dieser Klassen

kombiniert, um eine Vorlage für das aktuelle Objekt zu formen. Jedes Objekt füllt die einzelnen Variablen dann mit den Informationen, die seiner Situation entsprechen.

Methoden arbeiten auf ganz ähnliche Art: Neue Objekte haben Zugriff auf alle Methodennamen der eigenen Klasse und deren Superklassen in der Hierarchie. Dies wird allerdings dynamisch festgelegt, wenn eine Methode in einem laufenden Programm ausgeführt wird. Wenn Sie die Methode eines bestimmten Objekts aufrufen, dann prüft der Java-Interpreter zunächst die Klasse des Objekts, ob sich die Methode hier befindet. Wenn er die Methode nicht findet, dann sucht er nach dieser in der Superklasse. Das geht so lange weiter, bis er die Definition der Methode gefunden hat. Die Abbildung 2.5 illustriert dies.

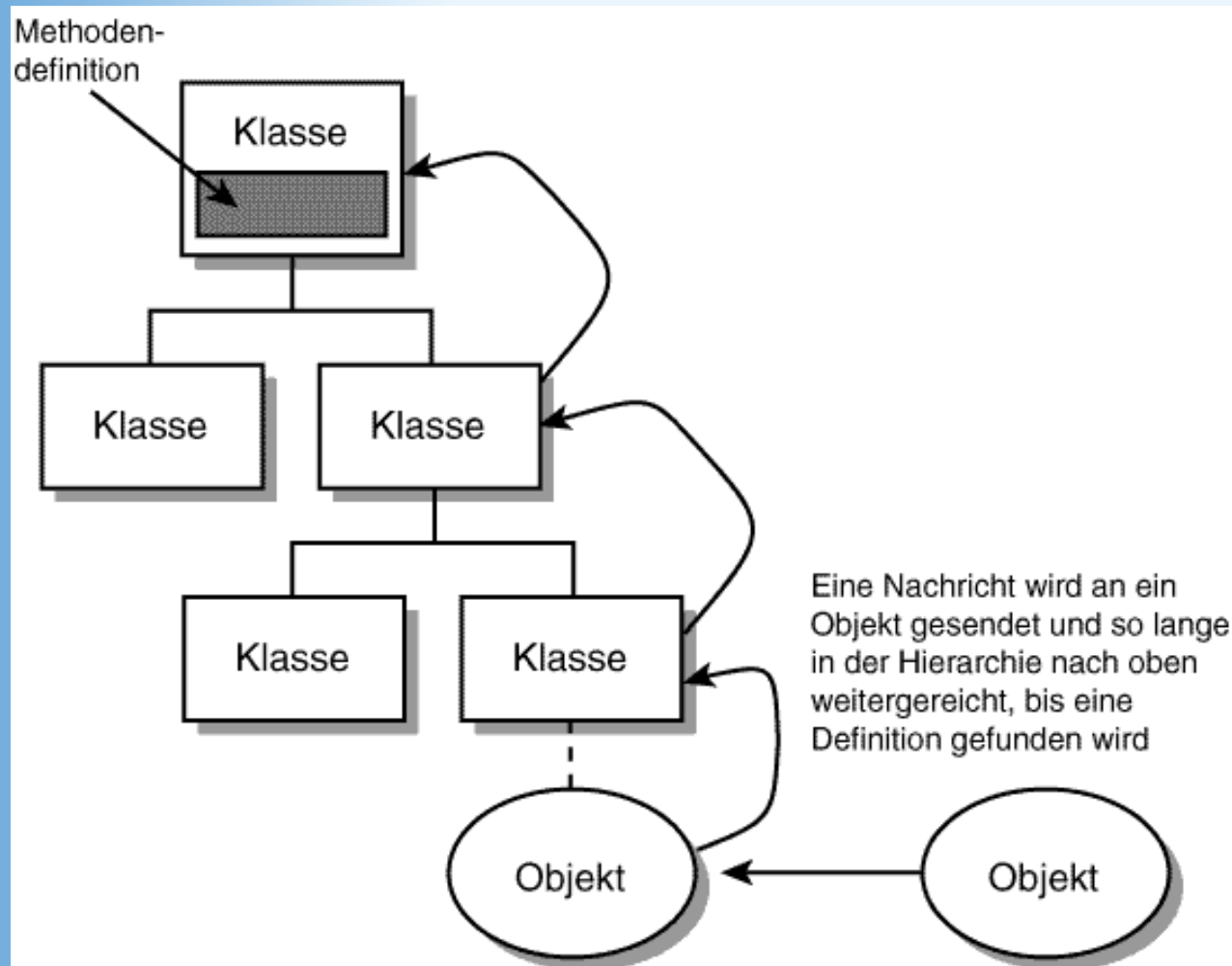


Abbildung 2.5:
So werden Methoden in der Klassenhierarchie geortet

Die Dinge werden komplizierter, wenn eine Subklasse eine Methode definiert, die denselben Namen, denselben Rückgabotyp und dieselben Parameter hat wie eine Methode in einer Superklasse. In diesem Fall wird die Methodendefinition, die als erstes gefunden wird (ausgehend vom unteren Ende der Hierarchie nach oben), verwendet. Aus diesem Grund können Sie in einer Subklasse eine Methode erstellen, die verhindert, daß eine Methode in einer Superklasse ausgeführt wird. Dazu geben Sie einer Methode denselben Namen, denselben Rückgabotyp und dieselben Argumente, wie sie die Methode in der Superklasse hat. Dieses Vorgehen wird als Überschreiben bezeichnet (siehe Abbildung 2.6).

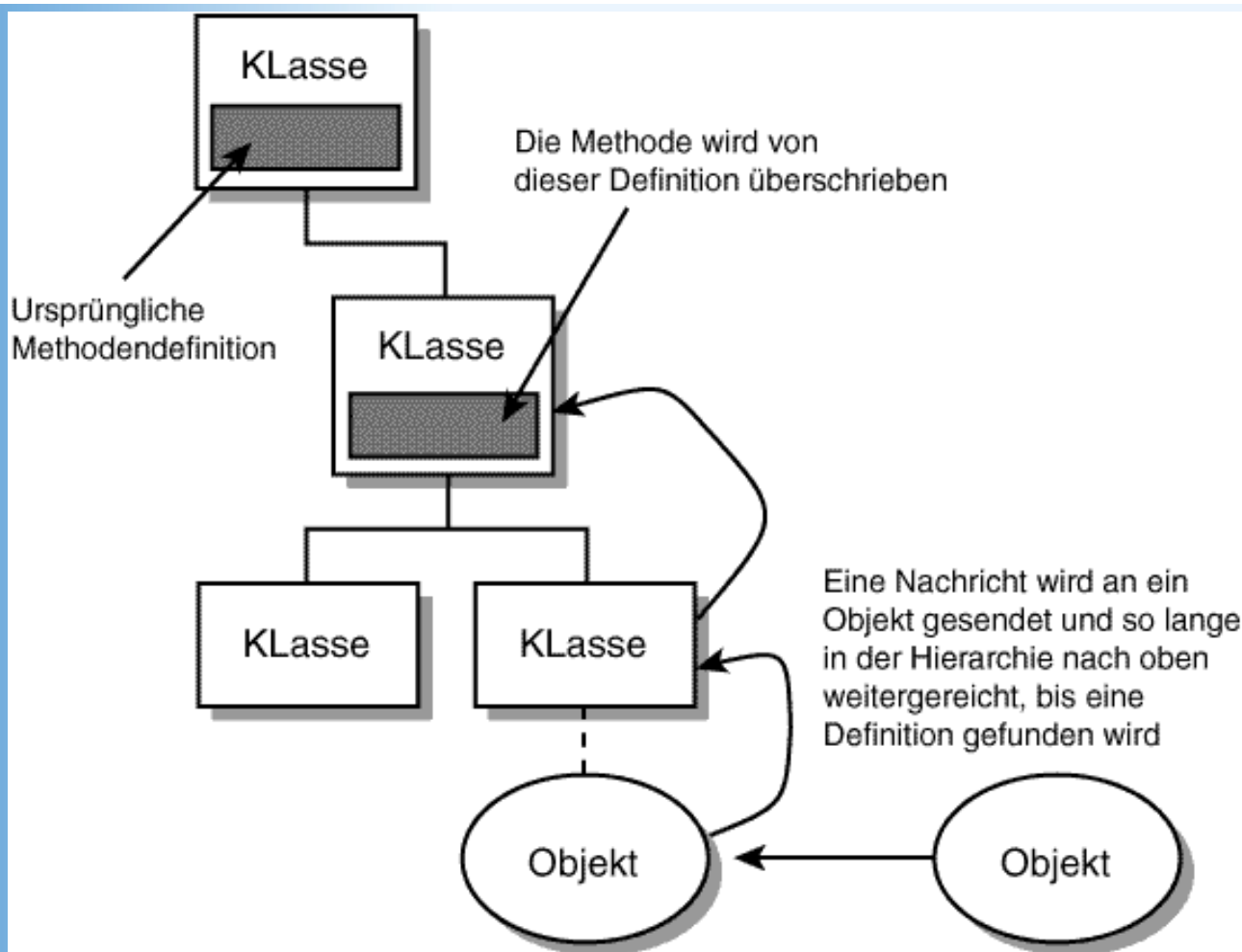


Abbildung 2.6:
Methoden überschreiben

Einfach- und Mehrfachvererbung

Javas Form der Vererbung wird Einfachvererbung genannt, da jede Java-Klasse nur eine Superklasse haben kann (allerdings kann jede Superklasse beliebig viele Subklassen haben).

In anderen objektorientierten Programmiersprachen, wie z.B. C++, können Klassen mehr als eine Superklasse haben, und sie erben natürlich die Variablen und Methoden aus allen Superklassen. Dies wird als Mehrfachvererbung bezeichnet und bietet die Möglichkeit, Klassen zu erzeugen, die nahezu alle denkbaren Verhaltensweisen und Attribute beinhalten. Allerdings werden dadurch die Definition von Klassen und der Code, der für deren Erstellung benötigt wird, bedeutend komplizierter. Java vereinfacht die Vererbung, indem es nur die Einfachvererbung zulässt.

Schnittstellen

Durch die Einfachvererbung ist die Beziehung zwischen Klassen und die Funktionalität, die diese Klassen implementieren, einfacher zu verstehen und zu entwerfen. Allerdings kann dies auch einschränkend sein - besonders dann, wenn Sie ähnliche Verhaltensweisen in verschiedenen Zweigen der Klassenhierarchie duplizieren müssen. Java löst das Problem von gemeinsam genutzten Verhaltensweisen durch Schnittstellen.



Eine *Schnittstelle* (engl. Interface) ist eine Sammlung von Methoden, die benannt aber nicht implementiert sind. Dadurch wird angezeigt, daß eine Klasse neben dem aus der Superklasse geerbten Verhalten noch zusätzliche Verhaltensweisen hat.

Eine Klasse kann beliebig viele Schnittstellen implementieren. Durch die Implementierung einer Schnittstelle wird die Klasse gezwungen die Methoden zu implementieren, deren Namen von der Schnittstelle definiert wurden. Wenn zwei sehr

unterschiedliche Klassen dieselbe Schnittstelle implementieren, können beide auf Aufrufe der Methoden, die in der Schnittstelle definiert sind, reagieren. Allerdings kann die Reaktion auf diese Methodenaufrufe bei den einzelnen Klassen total unterschiedlich sein.

Pakete

Pakete (engl. Packages) sind eine Möglichkeit, um verwandte Klassen und Schnittstellen zu gruppieren. Pakete ermöglichen es, daß Gruppen von Klassen nur bei Bedarf verfügbar sind. Zusätzlich beseitigen sie mögliche Namenskonflikte zwischen Klassen in unterschiedlichen Gruppen von Klassen.

Zum jetzigen Zeitpunkt gibt es nur ein paar wenige Dinge, die Sie über Pakete wissen müssen:

- Die Klassenbibliotheken von Java befinden sich in einem Paket, das `java` heißt. Für die Klassen in dem Paket `java` wird garantiert, daß sie in jeder Implementierung von Java zur Verfügung stehen. Dies sind die einzigen Klassen, für die die Garantie besteht, daß sie in unterschiedlichen Implementierungen zur Verfügung stehen. Das Paket `java` beinhaltet kleinere Pakete, die spezielle Teile der Funktionalität der Sprache Java, wie z.B. Standard-Features, Dateiein- und ausgaben, Multimedia und viele andere Dinge, definieren. Klassen in anderen Paketen, wie z.B. `sun` oder `netscape`, stehen oft nur in bestimmten Implementierungen zur Verfügung.
- Standardmäßig haben Ihre Klassen nur Zugriff auf die Klassen in dem Paket `java.lang` (Standard-Features der Sprache). Um Klassen aus irgendeinem anderen Paket zu verwenden, müssen Sie sich direkt auf diese beziehen oder sie in Ihren Quelltext importieren.
- Um sich auf eine Klasse in einem Paket zu beziehen, müssen Sie alle Pakete, in denen sich die Klasse befindet, angeben. Die einzelnen Elemente müssen dabei durch Punkte (.) voneinander getrennt werden. Nehmen Sie als Beispiel die Klasse `Color`, die sich in dem Paket `awt` befindet. Dieses Paket befindet sich seinerseits im Paket `java`. Um sich nun auf die Klasse `Color` in Ihren Programmen zu beziehen, können Sie die folgende Notation verwenden: `java.awt.Color`.

Eine Subklasse erstellen

Als abschließendes Projekt für heute werden Sie eine Subklasse einer anderen Klasse erstellen und einige Methoden überschreiben. Sie werden auch ein besseres Verständnis dafür bekommen, wie Pakete funktionieren.

Wenn Sie mit der Programmierung in Java beginnen, werden Sie Klassen zur Erstellung von Applets ableiten. Die Erstellung von Applets unterscheidet sich von der von Applikationen. Java-Applets werden als Teil einer Webseite ausgeführt. Aus diesem Grund gibt es besondere Regeln für deren Verhalten. Wegen dieser speziellen Regeln für Applets ist die Erstellung eines einfachen Applets komplizierter als die einer einfachen Applikation.

Alle Applets sind Subklassen der Klasse `Applet` (diese ist Bestandteil des Paketes `java.applet`). Indem Sie die Klasse `Applet` ableiten, erhalten Sie automatisch alle Verhaltensweisen und Attribute, die es einem Programm ermöglichen, als Teil einer Webseite zu laufen.

In diesem Beispiel erstellen Sie ein Applet, das der Applikation `HelloDan` von gestern ähnelt. Zu Beginn dieses Beispiels erzeugen Sie die Klassendefinition. Starten Sie Ihren Texteditor, und geben Sie die folgenden Anweisungen ein:

```
public class Palindrome extends java.applet.Applet {
    // mehr in Kürze
}
```

Dies definiert eine Klasse namens `Palindrome`. Die Anweisungen ähneln der Art und Weise, wie Sie die `HelloDan`-Applikation am ersten Tag erstellt haben. Eine Neuerung stellt der Text `extends java.applet.Applet` dar.

Über die Anweisung `extends` wird festgelegt, daß eine Klasse eine Subklasse einer anderen ist. Die Klasse `Palindrome` ist demnach eine Subklasse der Klasse `Applet`, die Teil des Paketes `java.applet` ist. Um in einem Programm die Beziehung zwischen den zwei Klassen deutlich zu machen, wird die Anweisung `extends java.applet.Applet` verwendet.



Da sich die Klasse `Applet` in dem Paket `java.applet` befindet, haben Sie nicht automatisch Zugriff auf diese Klasse. Aus diesem Grund müssen Sie sich explizit auf diese über den Paket- und Klassennamen beziehen. Die einzigen Klassen, die Sie ohne Angabe der Paketzugehörigkeit verwenden können, sind die in dem Paket `java.lang`.

Ein weiteres neues Element im class-Statement ist das Schlüsselwort `public`. Dieses Schlüsselwort zeigt an, daß andere Klassen auf Ihre Klasse zugreifen können, wenn sie diese benötigen. Im Normalfall müssen Sie eine Klasse nur dann als `public` deklarieren, wenn Sie wollen, daß sie von anderen Klassen in Ihrem Java-Programm verwendet werden kann. Alle Applets müssen allerdings `public` sein.

Eine Klassendefinition mit nichts außer dem Kommentar `//` mehr in Kürze darin ergibt nicht viel Sinn - sie fügt ihrer Superklasse weder etwas hinzu, noch überschreibt sie Methoden oder Variablen der Superklasse. Um die `Palindrome`-Klasse gegenüber ihrer Superklasse zu verändern, löschen Sie die Kommentarzeile `//` mehr in Kürze und fügen neue Anweisungen, beginnend mit der folgenden, hinzu:

```
Font f = new Font("TimesRoman", Font.BOLD, 36);
```

Diese Anweisung erfüllt zwei Aufgaben:

- Es wird ein `Font`-Objekt mit dem Namen `f` erzeugt. `Font` als Teil des `java.awt`-Paketes wird verwendet, um Bildschirmschriften zu repräsentieren. Mit einem solchen Objekt kann man eine Schrift anzeigen, die sich von der standardmäßig in Applets benutzten Schrift unterscheidet.
- Dem `Font`-Objekt wird die Schrift `Times Roman`, fett in `36`-Punkt zugewiesen. Die `new`-Anweisung erzeugt das neue `Font`-Objekt mit den in den Klammern angegebenen Werten. Das neu erzeugte Objekt wird anschließend der Variablen `f` zugewiesen.

Indem Sie eine Instanzvariable erstellen, die dieses `Font`-Objekt aufnimmt, machen Sie es für alle Methoden in Ihrer Klasse verfügbar. Der nächste Schritt in dem `Palindrome` -Projekt ist die Erstellung von Methoden, die dieses Objekt nutzen.

Wenn Sie Applets schreiben, werden Sie gewöhnlich einige Methoden, die in der Superklasse `Applet` definiert sind, in Ihrem Applet überschreiben. Diese beinhalten Methoden, um das Applet vor der eigentlichen Ausführung zu initialisieren, das Applet zu starten, auf Mauseingaben zu reagieren und aufzuräumen, wenn das Applet beendet wird.

Eine dieser Methoden ist `paint()`, die sich um die Anzeige des Applets auf einer Webseite kümmert. Die `paint()`-Methode, die `Palindrome` erbt, tut überhaupt nichts - sie ist einfach eine leere Methode. Indem Sie `paint()` überschreiben, legen Sie fest, was im Applet-Fenster, während das Programm läuft, bei Bedarf ausgegeben werden soll. Fügen Sie eine leere Zeile unter der `Font`-Anweisung ein, und geben Sie den folgenden Code ein:

```
public void paint(Graphics screen) {
    screen.setFont(f);
    screen.setColor(Color.red);
    screen.drawString("Go hang a salami, I'm a lasagna hog.", 5, 40);
}
```

Die `paint()`-Methode ist wie das Applet selbst als `public` deklariert. Hier allerdings aus einem anderen Grund: `paint()` muß in diesem Fall `public` sein, da die Methode, die es hier überschreibt, auch `public` ist. Eine `public`-Methode einer Superklasse kann nur von einer `public`-Methode überschrieben werden. Andernfalls wird das Java-Programm nicht kompiliert.

Die `paint()`-Methode besitzt ein einziges Argument: eine Instanz der Klasse `Graphics` mit dem Namen `screen`. Die Klasse `Graphics` stellt die Verhaltensweisen zur Darstellung von Schriften, Farben, zum Zeichnen von Linien und anderen Formen zur Verfügung. Sie werden während der zweiten Woche, wenn Sie diverse weitere Applets erstellen, mehr über die Klasse `Graphics` lernen.

In der `paint()`-Methode haben Sie drei Dinge getan:

- Sie haben dem `Graphics`-Objekt mitgeteilt, daß sich die Schrift zur Anzeige von Text in der Instanzvariablen `f` befindet.
- Sie haben dem `Graphics`-Objekt mitgeteilt, daß die Farbe für Text-Ausgaben und andere Zeichenoperationen eine Instanz der Klasse `Color` für die Farbe rot ist.
- Schließlich haben Sie den Text »Go hang a salami, I'm a lasagna hog« auf dem Bildschirm bei den (x, y)-Koordinaten 5, 25 ausgegeben. Der String wird in der festgelegten Schrift und Farbe angezeigt.

Das Applet sieht bis jetzt folgendermaßen aus:

```
public class Palindrome extends java.applet.Applet {
    Font f = new Font("TimesRoman", Font.BOLD, 36);
```



```

    public void paint(Graphics screen) {
        screen.setFont(f);
        screen.setColor(Color.red);
        screen.drawString("Go hang a salami, I'm a lasagna hog.", 5, 40);
    }
}

```

Sie haben vielleicht bemerkt, daß an diesem Punkt des Beispiels noch etwas fehlt. Wenn Sie nämlich die Datei gespeichert und versucht haben, diese zu kompilieren, dann werden Sie eine Reihe von Fehlermeldungen wie die folgende erhalten haben:

Palindrome.java:2: Class Font not found in type declaration.p

Diese treten auf, da die Klassen Graphics, Font und Color Teil des java.awt-Paketes sind. Und dieses Paket ist nicht standardmäßig verfügbar. Auf die Applet-Klasse haben Sie sich direkt in der ersten Zeile der Klassendefinition bezogen, indem Sie deren vollen Paketnamen (java.applet.Applet) angegeben haben. Im übrigen Programm haben Sie Klassen ohne deren Paketnamen verwendet.

Es gibt zwei Möglichkeiten, dieses Problem zu lösen:

- Beziehen Sie sich auf alle externen Klassen über deren vollen Paketnamen, wie z.B. java.awt.Graphics, java.awt.Font und java.awt.Color.
- Verwenden Sie eine import-Anweisung am Anfang des Programms, um ein oder mehrere Pakete und Klassen in dem Programm verfügbar zu machen.

Welche der beiden Lösungen Sie verwenden, bleibt Ihrer Wahl überlassen. Wenn Sie sich allerdings häufig auf eine Klasse in einem anderen Paket beziehen, dann werden Sie wahrscheinlich die import-Anweisung verwenden wollen, um den Tippaufwand zu reduzieren.

In diesem Beispiel werden wir die letztere Variante verwenden.

Um die Klassen zu importieren, fügen Sie die folgenden drei Anweisungen über der Anweisung public class Palindrome ein:

```

import java.awt.Graphics;
import java.awt.Font;
import java.awt.Color;

```



Sie können auch ein komplettes Paket importieren, indem Sie ein Sternchen (*) anstelle eines konkreten Klassennamens verwenden. Um z.B. alle Klassen des java.awt- Pakets zu importieren, verwenden Sie das folgende Statement:

```
import java.awt.*;
```

Nun, da die richtigen Klassen in Ihr Programm importiert wurden, sollte sich Palindrome.java problemlos in eine .class-Datei kompilieren lassen. In Listing 2.4 finden Sie zu Vergleichszwecken die endgültige Version.

Listing 2.4: Die endgültige Version von Palindrome.java

```

1: import java.awt.Graphics;
2: import java.awt.Font;
3: import java.awt.Color;
4:
5: public class Palindrome extends java.applet.Applet {
6:     Font f = new Font("TimesRoman", Font.BOLD, 36);
7:
8:     public void paint(Graphics screen) {
9:         screen.setFont(f);
10:        screen.setColor(Color.red);
11:        screen.drawString("Go hang a salami, I'm a lasagna hog.", 5, 40);
12:    }
13: }

```

Speichern Sie diese Datei als `Palindrome.java`. Diese Quelldatei kann auf dieselbe Art und Weise kompiliert werden wie die Java-Applikationen, die Sie bisher erstellt haben. Um das Programm auszuführen, müssen Sie allerdings eine Webseite erzeugen und es darauf einfügen.

Viele Programme zur Entwicklung von Webseiten, wie z.B. Claris Home Page, Macromedia Dreamweaver oder Microsoft FrontPage, ermöglichen es, ein Java-Applet auf einer Webseite einzufügen.

Wenn Sie keines dieser Tools zur Verfügung haben, können Sie eine einfache Webseite mit den entsprechenden Features zur Integration von Java-Applets direkt über die Seitenbeschreibungssprache HTML erstellen.



Obwohl einige der HTML-Features, die in Bezug zu Java stehen, in diesem Buch beschrieben werden, ist die Entwicklung von Webseiten mit HTML außerhalb des Rahmens dieses Buches. Wenn Sie an HTML interessiert sind, dann finden Sie in dem Buch [*HTML 4 in 14 Tagen*](#) von Laura Lemay (ISBN 3-8272-2019-X), erschienen bei Sams, eine umfangreiche und leichtverständliche Anleitung.

Um eine HTML-Seite zu erstellen, die das Palindrome-Applet aufnehmen kann, öffnen Sie denselben Texteditor, den Sie auch zur Erstellung Ihrer Java-Programme verwenden, und erzeugen ein neues Dokument.

Geben Sie den Text aus Listing 2.5 ein, und speichern Sie die Datei als `Palindrome.html` in demselben Ordner, in dem sich auch die Dateien `Palindrome.java` und `Palindrome.class` befinden. Wenn Sie Windows 95 verwenden, schließen Sie den Namen beim Abspeichern in Anführungszeichen ein, um sicherzugehen, daß die Erweiterung `.txt` nicht hinzugefügt wird.

Listing 2.5: Die Webseite `Palindrome.html`

```
1: <APPLET CODE="Palindrome.class" WIDTH=600 HEIGHT=100>
2: </APPLET>
```

Sie lernen über das HTML-Tag `<APPLET>` an anderer Stelle in diesem Buch mehr. Zwei Dinge sollen hier allerdings angemerkt sein:

- Das Attribut `CODE` gibt den Namen der Klasse an, die das Applet beinhaltet - in diesem Beispiel `Palindrome.class`.
- Die Attribute `WIDTH` und `HEIGHT` legen fest, wie groß das Applet-Fenster auf der Webseite in Pixeln sein wird. In diesem Beispiel wird das Fenster 600 Pixel breit und 100 Pixel hoch sein.

Um dieses Applet sehen zu können, benötigen Sie einen Web-Browser, der Java-Applets ausführen kann oder das Tool `appletviewer` aus dem JDK.



Alle Applets in diesem Buch verwenden, sofern nicht anders angegeben, nur Features von Java 1.0.2, so daß die Applets mit allen Browsern, die Java unterstützen, angezeigt werden können. Applikationen werden die Features von Java 1.2 verwenden, da diese direkt mit dem Java 1.2 Interpreter ausgeführt werden können.

Um die Webseite `Palindrome.html` in einem Browser zu öffnen, verwenden Sie ein Menükommando zum Öffnen lokaler Dateien anstelle von Seiten aus dem Web. Im Netscape Navigator 4.04 ist dies das Kommando `Datei | Seite öffnen | Datei wählen`.

Um eine Seite mit dem `appletviewer`-Tool des JDK zu öffnen, wechseln Sie an der MS-DOS-Eingabeaufforderung in das Verzeichnis, das die Datei `Palindrome.html` beinhaltet, und geben das folgende Kommando ein:

```
appletviewer Palindrome.html
```

Anders als ein Web Browser zeigt der Appletviewer nur das Applet (oder die Applets) an, die sich auf der Webseite befinden. Alles andere, was sich auf der Seite befindet, verarbeitet er nicht.

Die Abbildung 2.7 zeigt das Applet in dem Appletviewer.

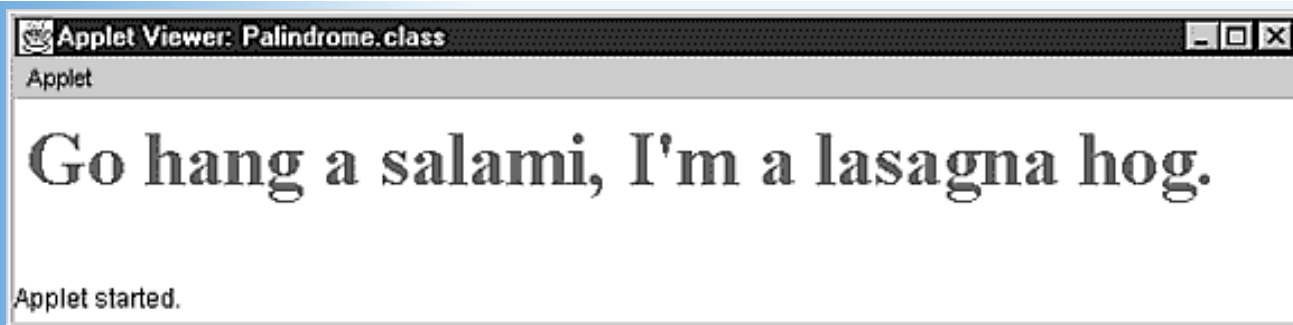


Abbildung 2.7:
Das Palindrome-Applet im Appletviewer



Wenn Sie noch nicht damit vertraut sind, was ein Palindrom ist, dann sehen Sie sich einmal die Abbildung 2.7 an, und lesen Sie den Satz »Go hang a salami, I'm a lasagna hog« einmal rückwärts. Palindrome sind Wörter oder Ausdrücke, die sich von vorne wie von hinten gleich lesen - wenn Sie alle Leer- und Satzzeichen vernachlässigen. Die Wörter »Otto« und »Rentner« wären weitere Beispiele für Palindrome. Im Web finden Sie unter der URL <http://www.tsoft.net/~derf/palindrome.html>, Neil/Fred's Gigantic List of Palindromes, eine umfangreiche Liste von Palindromen.

Zusammenfassung

Wenn dies Ihre erste Begegnung mit der objektorientierten Programmierung war, dann haben Sie vielleicht noch eine weitere Gemeinsamkeit mit Bier festgestellt.

Die objektorientierte Programmierung ist auch in der Lage, Sie schwindelig und verwirrt zu machen und vielleicht sogar einen Brechreiz auszulösen.

Wenn Ihnen der Stoff des heutigen Tages theoretisch und erschlagend erschien, dann sollten Sie nicht beunruhigt sein. Sie werden die objektorientierten Techniken im Rest des Buches verwenden, und diese werden Ihnen um so vertrauter werden, je mehr Erfahrung Sie damit haben.

Eine der größten Hürden der objektorientierten Programmierung ist nicht notwendigerweise das Konzept selbst, sondern sind die verwendeten Bezeichnungen. In der OOP findet sich mehr Jargon und eine leicht bedrohlich klingende technische Sprache als in einer Folge von *AkteX*.

Um den Stoff des heutigen Tages zusammenzufassen, finden Sie im folgenden ein Glossar der Begriffe und Konzepte, die heute behandelt wurden:

Klasse:	Eine Vorlage für ein Objekt. Diese beinhaltet Variablen, um das Objekt zu beschreiben, und Methoden, um zu beschreiben, wie sich das Objekt verhält. Klassen können Variablen und Methoden von anderen Klassen erben.
Objekt:	Eine Instanz einer Klasse. Mehrere Objekte, die Instanzen derselben Klasse sind, haben Zugriff auf dieselben Methoden, aber oft unterschiedliche Werte für deren Instanzvariablen.
Instanz:	Dasselbe wie ein Objekt. Jedes Objekt ist eine Instanz einer Klasse.
Methode:	Eine Gruppe von Anweisungen in einer Klasse, die definieren, wie sich Objekte dieser Klasse verhalten werden. Methoden sind analog zu Funktionen in anderen Programmiersprachen. Im Unterschied zu Funktionen müssen sich Methoden immer innerhalb einer Klasse befinden.
Klassenmethode:	Eine Methode, die auf eine Klasse selbst angewendet wird und nicht auf eine bestimmte Instanz einer Klasse.
Instanzmethode:	Eine Methode, die auf Instanzen einer Klasse angewendet wird. Da Instanzmethoden wesentlich häufiger verwendet werden als Klassenmethoden, werden Sie auch einfach nur als Methoden bezeichnet.

Klassenvariable:	Eine Variable, die ein Attribut einer ganzen Klasse anstatt einer bestimmten Instanz einer Klasse beschreibt.
Instanzvariable:	Eine Variable, die ein Attribut einer Instanz einer Klasse beschreibt.
Schnittstelle:	Eine Beschreibung abstrakter Verhaltensweisen, die einzelne Klassen implementieren können.
Paket:	Eine Sammlung von Klassen und Schnittstellen. Klassen, die sich nicht in dem Paket java.lang befinden, müssen explizit importiert oder direkt über den vollen Paket- und Klassennamen angesprochen werden.
Subklasse:	Eine Klasse, die sich in der Klassenhierarchie weiter unten befindet als eine andere Klasse, ihre Superklasse. Das Erzeugen einer Klasse, die Features einer bestehenden Klasse erbt, wird oft auch als Ableiten bezeichnet.
Superklasse:	Eine Klasse, die sich in der Klassenhierarchie weiter oben befindet als eine oder mehrere andere Klasse(n), ihre Subklasse(n). Eine Klasse kann nur eine Superklasse direkt über sich haben. Diese Klasse kann aber ihrerseits wieder eine Superklasse haben usw. Zu einer Superklasse kann es beliebig viele Subklassen geben.

Fragen und Antworten

Frage:

Methoden sind doch eigentlich nichts anderes als Funktionen, die innerhalb von Klassen definiert wurden. Wenn sie wie Funktionen aussehen und sich auch wie Funktionen verhalten, warum nennt man Sie dann nicht Funktionen?

Antwort:

In einigen objektorientierten Programmiersprachen werden sie Funktionen genannt (in C++ werden sie als Elementfunktionen bezeichnet). Andere objektorientierte Programmiersprachen unterscheiden zwischen Funktionen innerhalb und außerhalb des Rumpfes einer Klasse oder eines Objekts, da bei diesen Sprachen die Unterscheidung zwischen den Begriffen wichtig ist zum Verständnis, wie die einzelnen Funktionen arbeiten. Da dieser Unterschied in anderen Sprachen relevant und der Begriff »Methode« in der objektorientierten Programmierung üblich ist, verwendet Java ihn auch.

Frage:

Was ist der Unterschied zwischen Instanzvariablen und -methoden und deren Gegenstücken Klassenvariablen und -methoden?

Antwort:

Nahezu alles, was Sie in Java programmieren, bezieht sich auf Instanzen (auch Objekte genannt) und nicht auf die Klassen selbst. Für manche Verhaltensweisen und Attribute ist es allerdings sinnvoller, sie in der Klasse selbst zu speichern als in einem Objekt. Um z.B. eine neue Instanz einer Klasse zu erzeugen, benötigen Sie eine Methode, die für die Klasse selbst definiert und verfügbar ist. Andernfalls würden Sie in ein Henne-Ei-Dilemma hineingeraten - Sie können kein Baby-Objekt erzeugen ohne ein Mama-Objekt, das über eine Methode zum Babymachen verfügt. Und kein Mama-Objekt kann existieren, ohne zuvor ein Baby gewesen zu sein.

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 1

Tag 3

Das Java-ABC

Wie Sie bereits gelernt haben, besteht ein Java-Programm aus Klassen und Objekten, diese sind aus Methoden und Variablen aufgebaut. Methoden wiederum bestehen aus Anweisungen und Ausdrücken, in denen sich Operatoren finden.

An dieser Stelle könnte sich bei Ihnen die Befürchtung breit machen, daß Java wie die verschachtelten russischen Puppen, die Matryoska ist. Es scheint so, daß jede dieser Puppen eine weitere kleinere Puppe beinhaltet, die genauso kompliziert und detailreich wie die größere Freundin ist.

Entspannen Sie sich ..., dieses Kapitel schiebt die großen Puppen beiseite, um die kleinsten Elemente der Java-Programmierung zu enthüllen. Sie lassen Klassen, Objekte und Methoden hinter sich und untersuchen die elementaren Dinge, die Sie in einer einzelnen Zeile Java-Code erreichen können.

Die folgenden Themen werden heute behandelt:

- Java-Anweisungen und -Ausdrücke
- Variablen und Datentypen
- Kommentare
- Literale
- Arithmetik
- Vergleiche
- Logische Operatoren



Da Java sehr eng an C und C++ angelehnt ist, wird ein großer Teil der Themen in diesem Kapitel Programmierern, die in diesen Sprachen versiert sind, bekannt sein. Wenn nötig, werden technische Hinweise wie dieser spezifische Unterschiede zwischen Java und den anderen Sprachen erklären.

Anweisungen und Ausdrücke

Alle Aufgaben, die Sie in einem Java-Programm ausführen wollen, können in eine Folge von Anweisungen aufgeteilt werden.



Eine *Anweisung* (engl. Statement) ist ein einzelner Befehl einer Programmiersprache, der dafür sorgt, daß etwas passiert.

Anweisungen stehen für eine einzelne Aktion, die in einem Java-Programm ausgeführt wird. Die folgenden Zeilen stellen allesamt einfache Java-Anweisungen dar:

```
int age = 30;
import java.awt.dnd;
System.out.println("You're not the boss of me!");
player.score = 41367;
```

Einige Anweisungen erzeugen einen Wert, wie das beim Addieren zweier Zahlen in einem Programm der Fall ist. Derartige Anweisungen werden als *Ausdrücke* bezeichnet.



Ein *Ausdruck* (engl. Expression) ist eine Anweisung, die als Ergebnis einen Wert produziert. Dieser Wert kann zur späteren Verwendung im Programm gespeichert, direkt in einer anderen Anweisung verwendet oder überhaupt nicht beachtet werden. Der Wert, den eine Anweisung erzeugt, wird *Rückgabewert* genannt.

Manche Ausdrücke erzeugen numerische Rückgabewerte, wie das beim Addieren zweier Zahlen in dem oben erwähnten Beispiel der Fall war. Andere erzeugen einen booleschen Wert - true oder false - oder sogar ein Java-Objekt. Diese werden etwas später am heutigen Tag besprochen.

Obwohl Java-Programme je eine Anweisung pro Zeile enthalten, sagt diese nicht aus, wo eine Anweisung beginnt und wo sie endet. Es handelt sich dabei lediglich um eine Layout-Konvention. Jede Anweisung wird mit einem Strichpunkt (;) abgeschlossen. Ein Programmierer kann mehr als eine Anweisung in einer Zeile anordnen, und das Programm wird trotzdem erfolgreich kompiliert werden, wie das im folgenden der Fall ist:

```
j.color = "lemon yellow"; j.hungry = false;
```

Anweisungen werden in Java mit einem Paar geschweifeter Klammern ({}) gruppiert. Eine Gruppe von Anweisungen, die sich in diesen Klammern befindet, wird als Block oder Blockanweisung bezeichnet. Sie werden darüber am Tag 5 mehr lernen.

Variablen und Datentypen

In der Jabberwock-Applikation, die Sie gestern erstellt haben, haben Sie Variablen verwendet, um bestimmte Informationen ablegen zu können.



Variablen sind Orte, an denen, während ein Programm läuft, Informationen gespeichert werden können. Der Wert der Variablen kann unter deren Namen von jedem Punkt im Programm aus geändert werden.

Um eine Variable zu erstellen, müssen Sie dieser einen Namen geben und festlegen, welchen Typ von Informationen sie speichern soll. Sie können einer Variablen auch bei der Erzeugung einen Wert zuweisen.

In Java gibt es drei Arten von Variablen: Instanzvariablen, Klassenvariablen und lokale Variablen.

Instanzvariablen werden, wie Sie gestern gelernt haben, zur Definition der Attribute eines Objekts verwendet. Klassenvariablen definieren die Attribute einer gesamten Klasse von Objekten und beziehen sich auf alle Instanzen einer Klasse.

Lokale Variablen werden innerhalb von Methodendefinitionen verwendet oder sogar in kleineren Blockanweisungen innerhalb von Methoden. Diese Variablen können nur verwendet werden, während die Methode oder der Block von dem Java-Interpreter ausgeführt wird. Die Existenz dieser Variablen endet anschließend.

Obwohl alle diese Variablen fast auf dieselbe Art erzeugt werden, werden Klassen- und Instanzvariablen anders verwendet als lokale Variablen. Sie lernen heute mehr über lokale Variablen. Instanz- und Klassenvariablen werden wir am Tag 4 durchnehmen.



Anders als andere Sprachen verfügt Java nicht über globale Variablen - Variablen, auf die überall in einem Programm zugegriffen werden kann. Instanz- und Klassenvariablen werden für den Informationsaustausch zwischen einzelnen Objekten verwendet, weshalb keine Notwendigkeit für globale Variablen besteht.

Variablen erstellen

Bevor Sie eine Variable in einem Java-Programm verwenden können, müssen Sie die Variable erst einmal erzeugen, indem Sie deren Namen und die Art der Information deklarieren, die die Variable speichern soll. Als erstes wird dabei die Art der Information angegeben, gefolgt von dem Namen der Variablen. Im Anschluß an diesen Absatz sehen Sie einige Beispiele für Variablendeklarationen:

```
int highScore;
String username;
boolean gameOver;
```



Sie lernen etwas später am heutigen Tag mehr über Variablentypen. Eventuell sind Sie aber schon mit den Typen, die in diesem Beispiel verwendet wurden, vertraut. Der Typ `int` repräsentiert Integer (ganze Zahlen), der Typ `boolean` wird für `true`-`false`-Werte verwendet, und `String` ist ein spezieller Variablentyp, der zum Speichern von Text verwendet wird.

Lokale Variablen können an jeder Stelle in einer Methode, wie jede andere Java-Anweisung auch, deklariert werden. Bevor sie allerdings verwendet werden können, müssen sie deklariert werden. Normalerweise werden Variablendeklarationen direkt im Anschluß an die Anweisung, die eine Methode deklariert, platziert.

Im folgenden Beispiel, werden drei Variablen am Beginn der `main()`-Methode des Programms deklariert:

```
public static void main (String arguments[] ) {
    int total;
    String reportTitle;
    boolean active;
}
```

Wenn Sie mehrere Variablen desselben Typs deklarieren, können Sie dies in einer einzigen Zeile. Dazu trennen Sie die einzelnen Variablennamen mit Kommas. Die folgende Anweisung erzeugt drei `String`-Variablen mit den Namen `street`, `city` und `state`:

```
String street, city, state;
```

Variablen kann bei deren Erstellung ein Wert zugewiesen werden. Dazu verwenden Sie das Gleichheitszeichen (`=`) gefolgt von dem Wert. Die folgenden Anweisungen erzeugen neue Variablen und weisen diesen Werte zu:

```
int zipcode = 90210;
String name = "Brandon";
boolean cheatedOnKelly = true;
int age = 28, height = 70, weight = 140;
```

Wie die letzte Anweisung bereits andeutet, können Sie mehreren Variablen desselben Typs Werte zuweisen, indem Sie sie mit Kommas voneinander trennen.

Lokalen Variablen müssen Werte zugewiesen werden, bevor sie in einem Programm verwendet werden können. Ansonsten kann das Programm nicht erfolgreich kompiliert werden. Aus diesem Grund ist es eine gute Angewohnheit, allen lokalen Variablen Initialisierungswerte zuzuweisen.

Instanz- und Klassenvariablen erhalten automatisch einen Initialisierungswert - abhängig davon, welchen Typ von Information diese aufnehmen sollen:

- Numerische Variablen: 0
- Zeichen-Variablen: `'\0'`
- Boolesche Variablen: `false`
- Objekt-Variablen: `null`

Variablen benennen

Variablennamen müssen in Java mit einem Buchstaben, einem Unterstrich (`_`) oder einem Dollarzeichen (`$`) beginnen. Sie dürfen nicht mit einer Ziffer starten. Nach dem ersten Zeichen können Variablennamen jede beliebige Kombination von

Buchstaben und Ziffern enthalten.



Java unterstützt auch den Unicode-Zeichensatz, der den Standardzeichensatz plus Tausende anderer Zeichen beinhaltet, um internationale Alphabete zu repräsentieren. Zeichen mit Akzenten und andere Symbole können in Variablennamen verwendet werden, solange diese über eine Unicode-Nummer oberhalb von 00C0 verfügen.

Wenn Sie eine Variable benennen und diese in einem Programm verwenden, dann ist es wichtig daran zu denken, daß Java die Groß-/Kleinschreibung beachtet. Das heißt die Verwendung von Groß- und Kleinbuchstaben muß konsistent sein. Aus diesem Grund kann es in einem Programm eine Variable X und eine andere Variable x geben - und eine rose ist keine Rose ist keine ROSE.

In den Programmen in diesem Buch, und auch außerhalb, werden Variablen oft mit aussagekräftigen Namen versehen, die aus mehreren miteinander verbundenen Wörtern bestehen. Um einzelne Worte leichter innerhalb des Namens erkennen zu können, wird die folgende Faustregel verwendet:

- Der erste Buchstabe eines Variablennamens ist klein.
- Jedes darauffolgende Wort in dem Namen beginnt mit einem Großbuchstaben.
- Alle anderen Buchstaben sind Kleinbuchstaben.

Die folgenden Variablendeklarationen entsprechen diesen Regeln:

```
Button loadFile;
int areaCode;
boolean playerSetNewHighScore;
```

Variablentypen

Neben dem Namen muß eine Variablendeklaration auch den Typ der Information, die in der Variablen gespeichert werden soll, beinhalten. Als Typ kann einer der folgenden verwendet werden:

- Einer der elementaren Datentypen
- Der Name einer Klasse oder Schnittstelle
- Ein Array

Sie lernen am Tag 5, wie Sie Arrays deklarieren und verwenden. Diese Lektion konzentriert sich auf Variablentypen.

Datentypen

Es gibt acht elementare Variablentypen zum Speichern von Integern, Fließkomma- zahlen, Zeilen und booleschen Werten. Diese werden oft auch als primitive Typen bezeichnet, da sie feste Bestandteile der Sprache und keine Objekte sind. Aus diesem Grund sind diese Typen bei der Anwendung effizienter. Diese Datentypen haben im Gegensatz zu einigen Datentypen in anderen Programmiersprachen, unabhängig von der Plattform oder dem Betriebssystem, dieselbe Größe und Charakteristik.

Vier der Datentypen können Integer-Werte speichern. Welchen Sie verwenden, hängt von der Größe des zu speichernden Integers ab (siehe auch Tabelle 3.1).

Tabelle 3.1: Integer-Typen

Typ	Größe	Wertebereich
byte	8 Bit	-128 bis 127
short	16 Bit	-32.768 bis 32.767
int	32 Bit	-2.147.483.648 bis 2.147.483.647
long	64 Bit	-9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807

Alle diese Typen sind vorzeichenbehaftet, d.h., sie können sowohl positive als auch negative Werte aufnehmen. Welchen Typ Sie für eine Variable verwenden, hängt von dem Wertebereich ab, den die Variable aufnehmen soll. Keine

Integer-Variable kann einen Wert, der zu groß oder zu klein für den zugewiesenen Variablentyp ist, verlässlich speichern. Sie sollten bei der Zuweisung des Typs Vorsicht walten lassen.

Einen anderen Typ Zahlen, die gespeichert werden können, stellen die Fließkomma- Zahlen dar, die die Typen float oder double haben. Fließkomma-Zahlen sind Zahlen mit einem Dezimalanteil. Der Typ float sollte für die meisten Benutzer ausreichend sein, da er jede beliebige Zahl zwischen 1.4E-45 bis 3.4E+38 verarbeiten kann.

Der Typ char wird für einzelne Zeichen, wie z.B. Buchstaben, Ziffern, Interpunktionszeichen und andere Symbole verwendet, da Java den Unicode-Zeichensatz unterstützt.

Der letzte der acht elementaren Datentypen ist boolean. Wie Sie bereits gelernt haben, speichern boolesche Variablen unter Java entweder den Wert true oder den Wert false.

All diese Variablentypen sind in Kleinbuchstaben definiert, und Sie müssen sie auch in dieser Form in Programmen verwenden. Es gibt Klassen, die denselben Namen wie einige dieser Datentypen besitzen, allerdings mit anderer Groß-/Kleinschreibung - z.B. Boolean und Char. Diese haben eine andere Funktionalität in einem Java-Programm, so daß Sie diese nicht im Austausch füreinander verwenden können. Sie werden morgen erfahren, wie Sie diese speziellen Klassen verwenden.

Klassentypen

Neben den acht elementaren Typen kann eine Variable eine Klasse als Typ haben, wie das in den folgenden Beispielen der Fall ist:

```
String lastName = "Walsh";
Color hair;
Jabberwock firstMonster;
```

Wenn eine Variable eine Klasse als Typ hat, dann bezieht sich diese Variable auf ein Objekt dieser Klasse oder einer ihrer Subklassen.

Das letzte Beispiel in der obigen Liste, Jabberwock firstMonster; erzeugt eine Variable mit dem Namen firstMonster, die sich auf ein Jabberwock-Objekt bezieht.

Einer Variablen den Typ einer Superklasse zuzuweisen, kann sinnvoll sein, wenn die Variable sich auf ein Objekt einer von vielen Subklassen beziehen kann. Nehmen Sie z.B. eine Klassenhierarchie mit der Superklasse Frucht und den drei Subklassen Apfel, Birne und Erdbeere. Wenn Sie eine Variable des Typs Frucht mit dem Namen meineLieblingsFrucht erzeugen, kann diese auf ein Apfel-, ein Birne- oder ein Erdbeere -Objekt zu verweisen.

Wenn Sie eine Variable vom Typ Object deklarieren, heißt das, daß diese Variable jedes beliebige Objekt aufnehmen kann.



In Java gibt es nichts, was der typedef-Anweisung aus C und C++ entspricht. Um neue Typen in Java zu deklarieren, müssen Sie eine Klasse deklarieren. Diese Klasse können Sie dann als Typ für Variablen verwenden.

Variablen Werte zuweisen

Sobald eine Variable deklariert wurde, kann ihr über den Zuweisungsoperator (das Gleichheitszeichen =) ein Wert zugewiesen werden. Im Anschluß an diesen Absatz finden Sie zwei Beispiele für Zuweisungsanweisungen:

```
idCode = 8675309;
snappyDresser = false;
```

Kommentare

Eine der wichtigsten Methoden, die Lesbarkeit eines Programms zu verbessern, sind Kommentare.



Kommentare sind Informationen, die in einem Programm einzig für den Nutzen eines menschlichen Betrachters eingefügt wurden, der versucht, herauszufinden, was das Programm tut. Der Java-Compiler ignoriert die Kommentare komplett, wenn er eine ausführbare Version der Java-Quelldatei erstellt.

Es gibt verschiedene Arten von Kommentaren, die Sie in Java-Programmen nach Ihrem eigenen Ermessen verwenden können.

Die erste Methode, einen Kommentar in einem Programm einzufügen, ist, dem Kommentartext zwei Schrägstriche (//) voranzustellen. Dadurch wird alles nach den Schrägstrichen bis zum Ende der Zeile zu einem Kommentar, wie in der folgenden Anweisung:

```
int creditHours = 3; // Bonusstunden für den Kurs festlegen
```

In diesem Beispiel wird alles, angefangen bei // bis zum Ende der Zeile, von dem Java-Compiler nicht beachtet.

Wenn Sie einen Kommentar einfügen wollen, der länger als eine Zeile ist, dann starten Sie den Kommentar mit der Zeichenfolge /* und beenden ihn mit */. Alles zwischen diesen beiden Begrenzern wird als Kommentar gesehen, wie in dem folgenden Beispiel:

```
/*Dieses Programm wurde spät nachts unter dem Einfluß von Antihistamin-
Medikamenten, deren Verfallsdatum abgelaufen war, und Soda-
Wasser aus dem Supermarkt geschrieben. Ich übernehme keine Garantie, weder explizi
t noch implizit, dafür, daß dieses Programm einen sinnvollen wie auch immer gearte
ten Zweck erfüllt. */
```

Der letzte Kommentartyp soll sowohl vom Computer als auch vom Menschen lesbar sein. Wenn Sie einen Kommentar mit der Zeichenfolge /** (anstelle von /*) einleiten und ihn mit der Zeichenfolge */ beenden, wird der Kommentar als offizielle Dokumentation für die Funktionsweise der Klasse und deren public-Methoden interpretiert.

Diese Art von Kommentar kann von Tools, wie javadoc, das sich in dem JDK befindet, gelesen werden. Das Programm javadoc verwendet diese offiziellen Kommentare, um eine Reihe von Webseiten zu erzeugen, die das Programm, seine Klassenhierarchie und die Methoden dokumentieren.

Die gesamte offizielle Dokumentation der Klassenbibliothek von Java ist das Ergebnis von javadoc-Komentaren. Sie können sich die Dokumentation von Java 1.2 im Web unter der folgenden Adresse ansehen:

<http://java.sun.com:80/products/jdk/1.2/docs>

Literale

Neben Variablen werden Sie in Java-Anweisungen auch Literale verwenden.



Literale sind Zahlen, Text oder andere Informationen, die direkt einen Wert darstellen.

Literal ist ein Begriff aus der Programmierung, der im wesentlichen aussagt, daß das, was Sie eingeben, auch das ist, was Sie erhalten. Die folgende Zuweisungsanweisung verwendet ein Literal:

```
int jahr = 1998;
```

Das Literal ist 1998, da es direkt den Integer-Wert 1998 darstellt. Zahlen, Zeichen und Strings sind alles Beispiele für Literale.

Obwohl die Bedeutung und Anwendung von in den meisten Fällen Literalen sehr intuitiv erscheint, verfügt Java über einige Sondertypen von Literalen, die unterschiedliche Arten von Zahlen, Zeichen, Strings und booleschen Werten repräsentieren.

Zahlen-Literale

Java besitzt sehr viele Zahlen-Literale. Die Zahl 4 ist z.B. ein Integer-Literal des Variablentyps `int`. Es können aber auch Variablen des Typs `byte` und `short` zugewiesen werden, da die Zahl klein genug ist, um in beide Typen zu passen. Ein Integer-Literal, das größer ist, als was der Typ `int` aufnehmen kann, wird automatisch als Typ `long` verarbeitet. Sie können auch angeben, daß ein Literal ein long-Integer sein soll, indem Sie den Buchstaben `l` (`L` oder `l`) der Zahl hinzufügen. Die folgende Anweisung speichert z.B. den Wert 4 in einem long-Integer:

```
long pennyTotal = 4L;
```

Um eine negative Zahl als Literal darzustellen, stellen Sie dem Literal ein Minuszeichen (-) voran (z.B. `-45`).

Wenn Sie ein Integer-Literal benötigen, das das Oktal-System verwendet, dann stellen Sie der Zahl eine 0 voran. Der Oktal-Zahl 777 entspricht z.B. das Literal 0777. Bei Hexadezimal-Integern werden den Literalen die Zeichen 0x vorangestellt, wie z.B. 0x12 oder 0xFF.



Das oktale und das hexadezimale Zahlensystem sind für viele fortgeschrittenere Programmieraufgaben sehr bequem. Allerdings ist es unwahrscheinlich, daß Anfänger diese benötigen werden. Das Oktal-System hat als Basis die 8, das heißt, daß dieses System je Stelle nur die Ziffern 0 bis 7 verwenden kann. Der Zahl 8 entspricht die Zahl 10 im Oktal-System (oder 010 als Java-Literal).

Das Hexadezimal-System verwendet hingegen als Basis die 16 und kann deshalb je Stelle die 16 Ziffern verwenden. Die Buchstaben A bis F repräsentieren die letzten sechs Ziffern, so daß 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F die ersten 16 Zahlen sind.

Das Oktal- und das Hexadezimal-System eignen sich für bestimmte Programmieraufgaben besser als das Dezimal-System. Wenn Sie schon einmal mit HTML die Hintergrundfarbe einer Webseite festgelegt haben, dann haben Sie Hexadezimal-Zahlen verwendet.

Fließkomma-Literale verwenden einen Punkt (.) als Dezimaltrennzeichen. Die folgende Anweisung weist einer `double`-Variablen mit einem Literal einen Wert zu:

```
double myGPA = 2.25;
```

Alle Fließkomma-Literale werden standardmäßig als `double` verarbeitet und nicht als `float`. Um ein `float`-Literal anzugeben, müssen Sie den Buchstaben `f` (`F` oder `f`) dem Literal anhängen, wie in dem folgenden Beispiel:

```
float piValue = 3.1415927F;
```

Sie können in Fließkomma-Literalen Exponenten verwenden, indem Sie den Buchstaben `e` oder `E`, gefolgt von dem Exponenten, angeben. Dieser kann auch eine negative Zahl sein. Die folgenden Anweisungen verwenden die Exponential Schreibweise:

```
double x = 12e22;
double y = 19E-95;
```

Boolesche Literale

Die booleschen Werte `true` und `false` sind auch Literale. Dies sind die einzigen beiden Werte, die Sie bei der Wertzuweisung zu einer Variablen vom Typ `boolean` oder bei der Anwendung von booleschen Werten in einer Anweisung oder anderswo verwenden können.

Wenn Sie andere Sprachen, wie z.B. C, verwendet haben, dann erwarten Sie eventuell, daß der Wert 1 `true` und der Wert 0 `false` entspricht. Dies trifft auf Java allerdings nicht zu - Sie müssen die Werte `true` oder `false` verwenden, um boolesche Werte anzugeben. Die folgende Anweisung weist einer `boolean`-Variablen einen Wert zu:

```
boolean toThineOwnSelf = true;
```

Beachten Sie bitte, daß das Literal `true` nicht in Anführungszeichen eingeschlossen ist. Wenn dem so wäre, würde der Java-Compiler annehmen, daß es sich um einen String von Zeichen handelt.

Zeichenliterale

Zeichenliterale werden durch ein einzelnes Zeichen, das von einfachen Anführungszeichen umgeben ist, wie z.B. 'a', '#' und '3', dargestellt. Sie sind eventuell mit dem ASCII-Zeichensatz vertraut, der 128 Zeichen, darunter Buchstaben, Ziffern, Interpunktionszeichen und andere Zeichen, die in bezug auf den Computer nützlich sind, beinhaltet. Java unterstützt über den 16-Bit-Unicode-Standard Tausende weiterer Zeichen.

Einige Zeichenliterale repräsentieren Zeichen, die nicht druckbar sind oder nicht über die Tastatur direkt eingegeben werden können. Die Tabelle 3.2 führt die Codes auf, die diese Sonderzeichen wie auch Zeichen aus dem Unicode-Zeichensatz repräsentieren. Der Buchstabe d in den Oktal-, Hex- und Unicode-Escape-Codes steht für eine Zahl oder eine Hexadezimalziffer (a-f oder A-F).

Tabelle 3.2: Escape-Codes für Sonderzeichen

Escape-Sequenz	Bedeutung
\n	Neue Zeile
\t	Tabulator (Tab)
\b	Rückschritt (Backspace)
\r	Wagenrücklauf (Carriage return)
\f	Seitenvorschub (Formfeed)
\\	Inverser Schrägstrich (Backslash)
\'	Einfache Anführungszeichen
\"	Doppelte Anführungszeichen
\d	Oktal
\xd	Hexadezimal
\ud	Unicode-Zeichen



C/C++-Programmierer sollten beachten, daß Java keine Codes für \a (Bell) und \v (vertikaler Tabulator) beinhaltet.

String-Literale

Die letzte Literalart, die Sie in Java verwenden können, repräsentiert Strings. Ein Java-String ist ein Objekt und kein elementarer Datentyp. Strings werden auch nicht in Arrays gespeichert, wie das in Sprachen wie C der Fall ist. Da String-Objekte echte Objekte in Java sind, stehen Methoden zur Kombination und Modifikation von Strings zur Verfügung sowie um festzustellen, ob zwei Strings denselben Wert haben.

String-Literale bestehen aus einer Reihe von Zeichen, die in zwei doppelte Anführungszeichen eingeschlossen sind, wie das in den folgenden Anweisungen der Fall ist:

```
String coAuthor = "Laura Lemay, killer of trees";
String password = "swordfish";
```

Strings können die Escape-Sequenzen aus Tabelle 3.2 enthalten, wie das auch im nächsten Beispiel gezeigt wird:

```
String example = "Socrates asked, \"Hemlock is poison?\"";
System.out.println("Bob Kemp\nOne on One Sports\n2 a.m. to 6 a.m.");
String title = "Teach Yourself Java in a 3-Day Weekend\u2122"
```

In dem letzten Beispiel hier erzeugt die Unicode-Escape-Sequenz \u2122 auf Systemen, die Unicode unterstützen, das Trademarksymbol (™).



Die meisten Windows-95-Anwender in englischsprachigen Ländern werden wahrscheinlich keine Unicode-Zeichen sehen, wenn sie Java-Programme ausführen. Obwohl Java die Übertragung von Unicode-Zeichen unterstützt, muß auch das System des Benutzers Unicode unterstützen, damit diese Zeichen angezeigt werden können. Die Unterstützung von Unicode bietet lediglich eine Möglichkeit zur Kodierung von Zeichen für Systeme, die den Standard unterstützen. Java 1.0.2 unterstützte lediglich den Teilzeichensatz *Latin* des gesamten Unicode-Zeichensatzes. Java 1.1 und 1.2 sind in der Lage, jedes beliebige Unicode-Zeichen darzustellen, das von einer Schrift auf dem Host unterstützt wird.

Mehr Informationen über Unicode finden Sie auf der Website des Unicode-Konsortiums unter <http://www.unicode.org/>.

Obwohl String-Literale in einem Programm auf ähnliche Art und Weise verwendet werden wie andere Literale, werden sie hinter den Kulissen anders verarbeitet.

Wenn ein String-Literal verwendet wird, speichert Java diesen Wert als ein String- Objekt. Sie müssen nicht explizit ein neues Objekt erzeugen, wie das bei der Arbeit mit anderen Objekten der Fall ist. Aus diesem Grund ist der Umgang mit diesen genauso einfach wie mit primitiven Datentypen. Strings sind in dieser Hinsicht ungewöhnlich - keiner der anderen primitiven Datentypen wird bei der Verwendung als Objekt gespeichert. Sie lernen am heutigen und morgigen Tag mehr über Strings und die String-Klasse.

Ausdrücke und Operatoren

Ein Ausdruck ist eine Anweisung, die einen Wert erzeugt. Mit die gebräuchlichsten Ausdrücke sind die mathematischen wie in dem folgenden Code-Beispiel:

```
int x = 3;
int y = 4;
int z = x * y;
```

Die letzte Anweisung in diesem Beispiel ist ein Ausdruck. Der Multiplikations-Operator `*` wird verwendet, um die Integer `x` und `y` miteinander zu multiplizieren. Der Ausdruck erzeugt das Ergebnis dieser Multiplikation. Dieses Ergebnis wird in dem Integer `z` gespeichert.

Der Wert, der von einem Ausdruck erzeugt wird, wird als Rückgabewert bezeichnet, wie Sie ja bereits gelernt haben. Dieser Wert kann einer Variablen zugewiesen und auf viele andere Arten in Ihren Java-Programmen verwendet werden.

Die meisten Ausdrücke in Java beinhalten Operatoren wie `*`.



Operatoren sind spezielle Symbole, die für mathematische Funktionen, bestimmte Zuweisungsarten und logische Vergleiche stehen.

Arithmetische Operatoren

Es gibt in Java fünf Operatoren für die elementare Arithmetik. Diese werden in Tabelle 3.3 aufgeführt.

Tabelle 3.3: Arithmetische Operatoren

Operator	Bedeutung	Beispiel
+	Addition	3 + 4
-	Subtraktion	5 - 7
*	Multiplikation	5 * 5
/	Division	14 / 7
%	Modulo	20 % 7

Jeder Operator benötigt zwei Operanden, einen auf jeder Seite. Der Subtraktions- Operator (-) kann auch dazu verwendet werden, einen einzelnen Operanden zu negieren, was der Multiplikation des Operanden mit -1 entspricht.

Eine Sache, die man bei Divisionen unbedingt beachten muß, ist die Art der Zahlen, die man dividiert. Wenn Sie das Ergebnis einer Division in einem Integer speichern, wird das Ergebnis zu einer ganzen Zahl gerundet, da der int-Typ keine Fließkomma- Zahlen aufnehmen kann. Das Ergebnis des Ausdrucks $31 / 9$ wäre 3, wenn es in einem Integer gespeichert wird.

Das Ergebnis der Modulo-Operation, die den %-Operator verwendet, ist der Rest einer Division. $31 \% 9$ würde 4 ergeben, da bei der Division von 31 durch 9 der Rest 4 übrigbleibt.

Beachten Sie bitte, daß die meisten arithmetischen Operationen, an denen ein Integer beteiligt ist, ein Ergebnis vom Typ int haben, unabhängig von dem Originaltyp der Operanden. Wenn Sie mit anderen Zahlen, wie z.B. Fließkomma-Zahlen oder long- Integern arbeiten, sollten Sie sicherstellen, daß die Operanden denselben Typ aufweisen, den das Ergebnis haben soll.

Das Listing 3.1 zeigt ein Beispiel für einfache Arithmetik in Java.

Listing 3.1: Die Quelldatei AmoebaMath.Java

```

1: class AmoebaMath {
2:     public static void main (String arguments[]) {
3:         int x = 6;
4:         short y = 4;
5:         float a = .12f;
6:
7:         System.out.println("You start with " + x + " pet amoebas.");
8:         System.out.println("\tTwo get married and their spouses move in.");
9:         x = x + 2;
10:        System.out.println("You now have " + x);
11:
12:        System.out.println("\tMitosis occurs, doubling the number of
13:        amoebas.");
14:        x = x * 2;
15:        System.out.println("You now have " + x);
16:
17:        System.out.println("\tThere's a fight. " + y + " amoebas move out.");
18:        x = x - y;
19:        System.out.println("You now have " + x);
20:
21:        System.out.println("\tParamecia attack! You lose one-third of the
22:        colony.");
23:        x = x - (x / 3);
24:        System.out.println("You end up with " + x + " pet amoebas.");
25:        System.out.println("Daily upkeep cost per amoeba: $" + a);
26:        System.out.println("Total daily cost: $" + (a * x));
27:    }
28: }
```

Wenn Sie diese Applikation ausführen, erhalten Sie die folgende Ausgabe:

```

You start with 6 pet amoebas.
    Two get married and their spouses move in.
You now have 8
    Mitosis occurs, doubling the number of amoebas.
You now have 16
    There's a fight. 4 amoebas move out.
You now have 12
    Paramecia attack! You lose one-third of the colony.
You end up with 8 pet amoebas.
Daily upkeep cost per amoeba: $0.12
```

Total daily cost: \$0.96

In dieser einfachen Java-Applikation werden in den Zeilen 3-5 drei Variablen mit Initialisierungswerten erzeugt: der Integer `x`, der short-Integer `y` und die Fließkomma- Zahl `a`. Da der Standardtyp für Fließkomma-Zahlen `double` ist, wird ein `f` an das Literal `.12` angehängt. Dies sagt aus, daß es sich um einen float handelt.

Der Rest des Programms verwendet arithmetische Operatoren, um die Population der Amöben-Kolonie (keine Sorge: während ich dieses Kapitel schrieb, wurde keine Amöbe verletzt) zu berechnen.

Dieses Programm verwendet auch die Methode `System.out.println()` in diversen Anweisungen. Die Methode `System.out.println()` wird in einer Applikation verwendet, um Strings oder andere Informationen auf dem Standardausgabegerät anzuzeigen - gewöhnlich der Bildschirm.

`System.out.println()` erwartet ein einziges Argument innerhalb der Klammern: einen String. Um mehr als eine Variable oder ein Literal als Argument für `println()` zu verwenden, können Sie mit dem `+`-Operator diese Elemente zu einem einzigen String verknüpfen.

Sie lernen über diese Verwendung des `+`-Operators später mehr.

Mehr über die Zuweisung

Die Zuweisung eines Wertes an eine Variable ist ein Ausdruck, da dies einen Wert erzeugt. Aus diesem Grund können Sie Zuweisungsanweisungen hintereinander schreiben, wie in dem folgenden Beispiel:

```
x = y = z = 7;
```

In dieser Anweisung haben am Ende alle drei Variablen den Wert 7.

Die rechte Seite eines Zuweisungsausdrucks wird immer vor der Zuweisung ausgewertet. Dies macht es möglich, Ausdrücke wie den folgenden zu verwenden:

```
int x = 5;
x = x + 2;
```

In dem Ausdruck `x = x + 2;` wird als erstes `x + 2` berechnet. Das Ergebnis dieser Berechnung - 7 - wird anschließend `x` zugewiesen.

Es ist eine ganz gewöhnliche Vorgehensweise in der Programmierung, den Wert einer Variablen durch einen Ausdruck zu verändern. Es gibt eine ganze Reihe von Operatoren, die ausschließlich in diesen Fällen verwendet werden.

Die Tabelle 3.4 zeigt diese Zuweisungsoperatoren und die Ausdrücke, denen sie von der Funktion her entsprechen.

Tabelle 3.4:
Zuweisungsoperatoren

Ausdruck	Bedeutung
<code>x += y</code>	<code>x = x + y</code>
<code>x -= y</code>	<code>x = x - y</code>
<code>x *= y</code>	<code>x = x * y</code>
<code>x /= y</code>	<code>x = x /</code>



Die Zuweisungsoperatoren sind von der Funktionsweise her äquivalent mit den längeren Zuweisungsausdrücken, die sie ersetzen. Wenn allerdings eine der Seiten Ihres Zuweisungsausdrucks Teil eines komplexen Ausdrucks ist, gibt es Fälle, in denen die Operatoren nicht äquivalent sind. In Zweifelsfällen sollten Sie einen Ausdruck vereinfachen, indem Sie mehrere Zuweisungsanweisungen verwenden anstatt der Zuweisungsoperatoren.

Inkrementieren und Dekrementieren

Eine weitere sehr häufig vorkommende Aufgabe ist es, zu einem Integer eins hinzuzuzählen oder eins abzuziehen. Es gibt für diese Ausdrücke spezielle Operatoren, die Inkrement- bzw. Dekrement-Operatoren genannt werden.



Eine Variable zu *inkrementieren* bedeutet, zu deren Wert eins hinzuzuzählen. Eine Variable zu *dekrementieren* bedeutet dagegen, von deren Wert eins abzuziehen.

Der Inkrement-Operator ist ++ und der Dekrement-Operator --. Diese Operatoren werden direkt nach oder direkt vor einen Variablennamen plziert, wie das im folgenden Beispiel der Fall ist:

```
int x = 7;
x = x++;
```

In diesem Beispiel inkrementiert die Anweisung `x = x++` die Variable `x` von 7 auf 8.

Die Inkrement- und Dekrement-Operatoren können vor oder nach dem Namen einer Variablen stehen. Dies beeinflusst den Wert von Ausdrücken, die diese Operatoren beinhalten.



Inkrement- und Dekrement-Operatoren werden als *Präfix-Operatoren* bezeichnet, wenn sie vor dem Namen der Variablen aufgeführt werden, und als *Postfix-Operatoren*, wenn sie sich hinter dem Variablennamen befinden.

In einem einfachen Ausdruck, wie z.B. `standards--`, ist es für das Ergebnis unerheblich, ob Sie einen Präfix- oder einen Postfix-Operator verwenden. Wenn Inkrement- oder Dekrement-Operatoren allerdings Teil größerer Ausdrücke sind, ist die Wahl zwischen Präfix- und Postfix-Operatoren wichtig.

Nehmen Sie die beiden folgenden Ausdrücke:

```
int x, y, z;
x = 42;
y = x++;
z = ++x;
```

Diese beiden Ausdrücke erzeugen unterschiedliche Ergebnisse aufgrund des Unterschieds zwischen der Präfix- und der Postfix-Operation. Wenn Sie Postfix-Operatoren wie in `y = x++` verwenden, erhält `y` den Wert von `x`, bevor dieser um eins inkrementiert wurde. Wenn Sie dagegen Präfix-Operatoren wie in `z = ++x` verwenden, wird `x` um eins inkrementiert, bevor der Wert `z` zugewiesen wird. Das Endergebnis dieses Beispiels ist, daß `y` den Wert 42 und `z` den Wert 44 hat. `x` selbst hat auch den Wert 44.

Für den Fall, daß Ihnen noch nicht ganz klar ist, was hier passiert, habe ich Ihnen im folgenden noch einmal das Beispiel aufgeführt. Diesmal allerdings mit Kommentaren, die jeden einzelnen Schritt beschreiben:

```
int x, y, z; // x, y, und z werden deklariert
x = 42;      // x wird der Wert 42 zugewiesen
y = x++;     // y wird der Wert von x (42) zugewiesen, bevor x inkrementiert wird
             // anschließend wird x auf 43 inkrementiert
z = ++x;     // x wird auf 44 inkrementiert, und z wird der Wert von x zugewiesen
```



Wie auch Zuweisungsoperatoren können Inkrement- und Dekrement-Operatoren unerwünschte Ergebnisse erzeugen, wenn diese in extrem komplexen Ausdrücken verwendet werden. Das Konzept »`x` wird `y` zugewiesen, bevor `x` inkrementiert wird« stimmt nicht ganz, da Java alles auf der rechten Seite eines Ausdrucks auswertet, bevor das Ergebnis der linken Seite zugewiesen wird. Java speichert einige Werte, bevor es einen Ausdruck verarbeitet, damit die

Postfix-Notation, wie in diesem Abschnitt beschrieben, funktionieren kann. Wenn Sie nicht die Ergebnisse erhalten, die Sie von einem komplexen Ausdruck mit Präfix- und Postfix-Operatoren erwarten, versuchen Sie den Ausdruck in mehrere Ausdrücke aufzuspalten, um ihn zu vereinfachen.

Vergleiche

Java besitzt diverse Operatoren, die bei Vergleichen von Variablen mit Variablen, Variablen mit Literalen oder anderen Informationsarten in einem Programm verwendet werden. Diese Operatoren werden in Ausdrücken verwendet, die boolesche Werte (true oder false) zurückgeben. Dies ist abhängig davon, ob der Vergleich aufgeht oder nicht. Die Tabelle 3.5 zeigt die einzelnen Vergleichsoperatoren.

Tabelle 3.5: Vergleichsoperatoren

Operator	Bedeutung	Beispiel
==	Gleich	x == 3
!=	Ungleich	x != 3
<	Kleiner als	x < 3
>	Größer als	x > 3
<=	Kleiner als oder gleich	x <= 3
>=	Größer als oder gleich	x >= 3

Die folgenden Beispiele zeigen die Verwendung eines Vergleichsoperators:

```
boolean hip;
int age = 31;
hip = age < 25;
```

Der Ausdruck `age < 25` ergibt entweder true oder false. Dies hängt von dem Wert des Integers `age` ab. Da `age` in diesem Beispiel den Wert 31 hat (was nicht kleiner als 25 ist), wird `hip` der boolesche Wert false zugewiesen.

Logische Operatoren

Ausdrücke, die boolesche Werte ergeben, wie das bei Vergleichen der Fall ist, können kombiniert werden, um komplexere Ausdrücke zu formen. Dies erreicht man mit logischen Operatoren. Diese Operatoren werden für die logischen Verknüpfungen AND (UND), OR (ODER), XOR (exklusives ODER) und NOT (logisches NICHT) verwendet.

Für AND-Verknüpfungen werden die Operatoren `&` und `&&` verwendet. Wenn zwei boolesche Ausdrücke mit `&` oder `&&` verknüpft werden, ergibt der kombinierte Ausdruck nur dann true, wenn beide Teilausdrücke true sind.

Nehmen Sie das folgende Beispiel, das direkt aus dem Film *Harold & Maude* stammt:

```
boolean unusual = (age < 21) & (girlfriendAge > 78);
```

Dieser Ausdruck kombiniert zwei Vergleichsausdrücke: `age < 21` und `girlfriendAge > 78`. Wenn beide Ausdrücke true ergeben, wird der Variablen `unusual` der Wert true zugewiesen, in allen anderen Fällen der Wert false.

Der Unterschied zwischen `&` und `&&` liegt in der Menge der Arbeit, die sich Java mit kombinierten Ausdrücken macht. Wenn `&` verwendet wird, werden immer die Ausdrücke auf beiden Seiten des `&` ausgewertet. Wenn dagegen `&&` verwendet wird und die linke Seite von `&&` false ergibt, wird der Ausdruck auf der rechten Seite nicht ausgewertet.

Für OR-Verknüpfungen werden die logischen Operatoren `|` oder `||` verwendet. Kombinierte Ausdrücke mit diesen Operatoren geben true zurück, wenn einer der beiden booleschen Ausdrücke true ergibt.

Nehmen Sie das von *Harold & Maude* inspirierte Beispiel:

```
boolean unusual = (suicideAttempts > 10) || (girlfriendAge > 78);
```

Dieser Ausdruck verknüpft zwei Vergleichsausdrücke: `suicideAttempts > 10` und `girlfriendAge > 78`. Wenn einer dieser Ausdrücke true ergibt, wird der Variablen `unusual` true zugewiesen. Nur wenn beide Ausdrücke false ergeben, wird `unusual` false zugewiesen.

Beachten Sie bitte, daß hier `||` anstelle von `|` verwendet wurde. Aus diesem Grund wird `unusual` true zugewiesen, wenn

`suicideAttempts > 10` true ergibt, und der zweite Ausdruck wird nicht ausgewertet.

Für die XOR-Operation gibt es nur einen Operator - $\wedge$. Ausdrücke mit diesem Operator ergeben nur dann true, wenn die booleschen Ausdrücke, die damit verknüpft werden, entgegengesetzte Werte haben. Wenn beide true oder beide false sind, ergibt der $\wedge$ - Ausdruck false.

Die NOT-Verknüpfung verwendet den logischen Operator `!`, gefolgt von einem einzelnen Ausdruck. Diese Verknüpfung kehrt den Wert eines booleschen Ausdrucks um, wie auch ein - (Minus) ein negatives oder positives Vorzeichen bei einer Zahl umkehrt.

Wenn z.B. der Ausdruck `age < 30` true zurückgibt, dann gibt `!(age < 30)` false zurück.

Diese logischen Operatoren erscheinen zunächst einmal völlig unlogisch, wenn man das erste Mal auf sie trifft. Sie werden in den folgenden Kapiteln, besonders an Tag 5, viele Gelegenheiten erhalten, mit diesen zu arbeiten.

Operatorpräzedenz

Sobald mehr als ein Operator in einem Ausdruck verwendet wird, verwendet Java eine definierte Präzedenz, um die Reihenfolge festzulegen, mit der die Operatoren ausgewertet werden. In vielen Fällen legt diese Präzedenz den Gesamtwert des Ausdrucks fest.

Nehmen Sie den folgenden Ausdruck als Beispiel:

```
y = 6 + 4 / 2;
```

Die Variable `y` erhält den Wert 5 oder den Wert 8, abhängig davon, welche arithmetische Operation zuerst ausgeführt wird. Wenn der Ausdruck `6 + 4` als erstes ausgewertet wird, hat `y` den Wert 5. Andernfalls erhält `y` den Wert 8.

Im allgemeinen gibt es folgende Reihenfolge, wobei der erste Eintrag der Liste die höchste Priorität besitzt:

- Inkrement- und Dekrement-Operationen
- Arithmetische Operationen
- Vergleiche
- Logische Operationen
- Zuweisungsausdrücke

Wenn zwei Operationen dieselbe Präzedenz besitzen, wird die auf der linken Seite in dem aktuellen Ausdruck vor der auf der rechten Seite ausgewertet. Tabelle 3.6 zeigt die Präzedenz der einzelnen Operatoren. Operatoren weiter oben in der Tabelle werden vor denen darunter ausgewertet.

Tabelle 3.6: Operatorpräzedenz

Operator	Anmerkung
<code>. [] ()</code>	Klammern (<code>()</code>) werden verwendet, um Ausdrücke zu gruppieren. Der Punkt (<code>.</code>) dient für den Zugriff auf Methoden und Variablen in Objekten und Klassen (wird morgen behandelt). Eckige Klammern (<code>[]</code>) kommen bei Arrays zum Einsatz (wird später in dieser Woche besprochen).
<code>++ - ! ~ instanceof</code>	Der Operator <code>instanceof</code> gibt true oder false zurück, abhängig davon, ob ein Objekt eine Instanz der angegebenen Klasse oder deren Subklassen (wird morgen besprochen) ist.
<code>new (typ)Ausdruck</code>	Mit dem <code>new</code> -Operator werden neue Instanzen von Klassen erzeugt. Die Klammern (<code>()</code>) dienen in diesem Fall dem Casting eines Wertes in einen anderen Typ (wird morgen behandelt).
<code>* / %</code>	Multiplikation, Division, Modulo
<code>+ -</code>	Addition, Subtraktion
<code><< >> >>></code>	Bitweiser Links- und Rechts-Shift
<code>< > <= >=</code>	Relationale Vergleiche
<code>== !=</code>	Gleichheit
<code>&</code>	AND
<code>^</code>	XOR
<code> </code>	OR

&&	Logisches AND
	Logisches OR
? :	Kurzform für if...then...else (wird an Tag 5 behandelt)
= += -= *= /= %= ^=	Verschiedene Zuweisungen
&= = <<= >>= >>>=	Weitere Zuweisungen

Lassen Sie uns nun zu dem Ausdruck $y = 6 + 4 / 2$ zurückkehren. Tabelle 3.7 zeigt, daß eine Division vor einer Addition ausgewertet wird, so daß y den Wert 8 haben wird.

Um die Reihenfolge, in der Ausdrücke ausgewertet werden, zu ändern, schließen Sie den Ausdruck, der zuerst ausgeführt werden soll, in Klammern ein. Sie können Klammernebenen ineinander verschachteln, um sicherzustellen, daß die Ausdrücke in der gewünschten Reihenfolge ausgeführt werden - der innerste geklammerte Ausdruck wird als erstes ausgeführt.

Der folgende Ausdruck ergibt den Wert 5:

```
y = (6 + 4) / 2
```

Das Ergebnis ist hier 5, da $6 + 4$ hier berechnet wird, bevor das Ergebnis durch 2 geteilt wird.

Klammern können auch nützlich sein, um die Lesbarkeit eines Ausdrucks zu erhöhen. Wenn Ihnen die Präzedenz eines Ausdrucks nicht sofort klar ist, dann kann das Hinzufügen von Klammern, um die geforderte Präzedenz zu erzwingen, den Ausdruck leichter verständlich machen.

String-Arithmetik

Wie zuvor schon erwähnt, führt der Operator `+` ein Doppelleben außerhalb der Welt der Mathematik. Er kann dazu verwendet werden, zwei oder mehrere Strings miteinander zu verketteten.

In vielen Beispielen haben Sie Anweisungen wie die folgende gesehen:

```
String firstName = "Raymond";
System.out.println("Everybody loves " + firstName);
```

Als Ergebnis der beiden Zeilen wird auf dem Bildschirm folgendes ausgegeben:

```
Everybody loves Raymond
```

Der Operator `+` kombiniert Strings, andere Objekte und Variablen zu einem einzigen String. In dem vorangegangenen Beispiel wird das Literal `Everybody loves` mit dem Wert des String-Objektes `firstName` verkettet.

Der Umgang mit dem Verkettungsoperator ist in Java einfach, da er alle Variablentypen und Objektwerte wie Strings behandelt. Sobald ein Teil einer Verkettung ein String oder ein String-Literal ist, werden alle Elemente der Operation wie Strings behandelt.

Zum Beispiel:

```
System.out.println(4 + " score " and " + 7 + " years ago.");
```

Diese Zeile erzeugt die Ausgabe `4 score and 7 years ago`, als ob die Integer-Literale 4 und 7 Strings wären.

Es gibt auch eine Kurzform, den Operator `+=`, um etwas an das Ende eines Strings anzuhängen. Nehmen Sie z.B. folgenden Ausdruck:

```
myName += " Jr.";
```

In diesem Beispiel wird dem Wert von `myName` (eventuell `Efrem Zimbalist`) am Ende der String `Jr.` hinzugefügt (was `Efrem Zimbalist Jr.` ergibt).

Zusammenfassung

Jeder, der eine Matryoska-Puppe zerlegt, wird ein bißchen enttäuscht sein, wenn er die kleinste Puppe der Gruppe erreicht. Idealerweise sollten es die Fortschritte in der Mikrotechnik den Künstlern ermöglichen, immer kleinere Puppen zu

erzeugen, bis einer die Schwelle zum subatomaren erreicht und als Gewinner feststeht.

Sie haben heute die kleinste Puppe von Java erreicht, es sollte aber kein Grund zur Enttäuschung sein. Anweisungen und Ausdrücke ermöglichen es Ihnen, mit der Erstellung effektiver Methoden zu beginnen, die wiederum effektive Objekte und Klasse ermöglichen.

Heute haben Sie gelernt, Variablen zu erstellen und diesen Werte zuzuweisen. Dazu haben Sie Literale, die numerische Werte, Zeichen und String-Werte repräsentierten, verwendet und mit Operatoren gearbeitet. Morgen werden Sie dieses Wissen verwenden, um Objekte für Java-Programme zu erstellen.

Um den heutigen Stoff zusammenzufassen, listet Tabelle 3.7 die Operatoren auf, die Sie heute kennengelernt haben.

Tabelle 3.7: Zusammenfassung der Operatoren

Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo
<	Kleiner als
>	Größer als
<=	Kleiner als oder gleich
>=	Größer als oder gleich
==	Gleich
!=	Nicht gleich
&&	Logisches AND
	Logisches OR
!	Logisches NOT
&	AND
	OR
^	XOR
=	Zuweisung
++	Inkrement
-	Dekrement
+=	Addieren und zuweisen
-=	Subtrahieren und zuweisen
*=	Multiplizieren und zuweisen
/=	Dividieren und zuweisen
%=	Modulo und zuweisen

Fragen und Antworten

Frage:

Was passiert, wenn man einen Integer-Wert einer Variablen zuweist und der Wert zu groß für die Variable ist?

Antwort:

Logischerweise werden Sie denken, daß die Variable in den nächstgrößeren Wert konvertiert wird. Allerdings ist dies genau das, was nicht passiert. Statt dessen tritt ein Überlauf auf. Dies ist eine Situation, in der eine Zahl von einer

Extremgröße in eine andere umkippt. Ein Beispiel für einen Überlauf wäre eine byte-Variable, die von dem Wert 127 (akzeptabler Wert) zu dem Wert 128 (nicht akzeptabel) springt. Der Wert der Variablen würde zu dem kleinsten zulässigen Wert (-128) umkippen und von dort aus fortfahren hochzuzählen. Überläufe können Sie in Programmen nicht auf die leichte Schulter nehmen. Aus diesem Grund sollten Sie Ihren Variablen reichlich Lebensraum für deren Datentyp geben.

Frage:

Warum gibt es in Java die Kurzformen für arithmetische Operationen und Zuweisungen? Es ist wirklich schwierig, Quelltexte mit diesen Operatoren zu lesen.

Antwort:

Die Syntax von Java basiert auf C++, das wiederum auf C basiert (schon wieder eine russische Puppe). C ist eine Sprache für Experten, die Mächtigkeit bei der Programmierung über die Lesbarkeit des Quellcodes stellt. Die Zuweisungsoperatoren sind eine Hinterlassenschaft dieser Priorität beim Design von C. Es besteht keine Notwendigkeit, diese in einem Programm zu verwenden, da es effektive Ersetzungsmöglichkeiten gibt. Wenn Sie es vorziehen, können Sie auf diese Operatoren ganz verzichten.

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).

Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 1

Tag 4

Arbeiten mit Objekten

Vor zwei Tagen wurde die objektorientierte Programmierung mit Bier verglichen, da der »Geschmack« gewöhnungsbedürftig ist und zur Desorientierung führen kann.

Im heutigen Kapitel können Sie nachschenken lassen. Während Sie in Java mit Objekten arbeiten, werden Sie mit dieser Art der Programmierung vertrauter werden.

Die meisten Dinge, die Sie mit dieser Sprache tun, tun Sie mit Objekten. Sie erzeugen Objekte, modifizieren sie, verschieben sie, verändern deren Variablen, rufen deren Methoden auf und kombinieren sie mit anderen Objekten. Sie werden Klassen entwickeln, Objekte dieser Klassen erzeugen und sie mit anderen Klassen und Objekten zusammen verwenden.

Sie werden heute sehr ausführlich mit Objekten arbeiten. Die folgenden Themen werden dabei behandelt:

- Erstellen von Objekten (auch Instanzen genannt)
- Testen und Ändern von Klassen- und Instanzvariablen in diesen Objekten
- Aufrufen von Methoden in einem Objekt
- Konvertieren von Objekten und anderen Datentypen von einer Klasse in eine andere

Erstellen neuer Objekte

Wenn Sie ein Java-Programm schreiben, definieren Sie verschiedene Klassen. Wie Sie am 2. Tag gelernt haben, dienen Klassen als Vorlagen für Objekte. Zum großen Teil benutzen Sie die Klassen lediglich, um Instanzen zu erstellen. Dann arbeiten Sie mit diesen Instanzen. In dieser Lektion lernen Sie, wie man ein neues Objekt aus einer Klasse erstellt.

Sie erinnern sich an die Zeichenketten aus der gestrigen Lektion? Sie haben gelernt, daß ein String-Literal - eine Reihe von Zeichen zwischen doppelten Anführungszeichen - eine neue Instanz der Klasse String mit dem Wert der jeweiligen Zeichenkette erzeugt.

Die String-Klasse ist in dieser Hinsicht ungewöhnlich. Es ist zwar eine Klasse, dennoch gibt es eine einfache Möglichkeit, anhand eines Literals Instanzen von dieser Klasse anzulegen. Für die anderen Klassen gibt es diese Abkürzung nicht. Um Instanzen dieser Klassen zu erstellen, müssen sie explizit mit dem Operator `new` angelegt werden.



Literale für Zahlen und Zeichen erstellen keine Objekte. Die primitiven Datentypen für Zahlen und Zeichen erstellen Zahlen und Zeichen, sind aber aus Effizienzgründen keine Objekte. Sie können sie in *Objekt-Wrapper* (Hüllklassen) verpacken, wenn Sie sie wie Objekte behandeln wollen (hierüber lernen Sie später mehr).

new

Um ein neues Objekt zu erstellen, benutzen Sie den Operator `new` mit dem Namen der Klasse, von der Sie eine Instanz anlegen wollen, und Klammern:

```
String teamName = new String();
```

```
Random randInfo = new Random();
Jabberwock j = new Jabberwock();
```

Die Klammern sind wichtig. Sie dürfen auf keinen Fall weggelassen werden. Die Klammern können leer bleiben. In diesem Fall wird ein ganz einfaches Objekt erstellt. Die Klammern können aber auch Argumente enthalten, die die Anfangswerte von Instanzvariablen oder andere Anfangsqualitäten des Objekts bestimmen:

```
GregorianCalendar date = new GregorianCalendar(64, 6, 6, 7, 30);
Point pt = new Point(0,0);
```

Zahl und Typ von Argumenten, die Sie mit `new` verwenden können, werden von der Klasse anhand einer speziellen Methode namens *Konstruktor* vorgegeben. Sie lernen später in dieser Woche noch alles über Konstruktor-Methoden. Wenn Sie versuchen eine Instanz einer Klasse über `new` mit der falschen Anzahl an Parametern zu erzeugen (oder Sie geben keine Parameter an, obwohl welche erwartet werden), tritt ein Fehler auf, sobald Sie versuchen, das Java-Programm zu kompilieren.

Hier nun ein Beispiel für die Erstellung einiger Objekte verschiedener Typen. Diese werden mit unterschiedlicher Anzahl und verschiedenen Typen von Argumenten erzeugt: Die Klasse `Random`, Teil des Paketes `java.util`, erzeugt Objekte, mit denen in einem Programm Zufallszahlen erzeugt werden können. Diese Objekte werden als Zufallsgeneratoren bezeichnet. Der Dezimalwert dieser Objekte liegt im Bereich von 0.0 bis 1.0. Zufallszahlen sind in Spielen und anderen Programmen, in denen ein Element der Unvorhersehbarkeit benötigt wird, enthalten.

Das `Random`-Objekt erzeugt nicht wirklich zufällig Zahlen. Statt dessen pickt es eine Zahl aus einer sehr großen Folge von Zahlen heraus. Dies wird auch als pseudozufällige Erzeugung bezeichnet und in vielen anderen Programmiersprachen verwendet. Um eine andere Zahl aus der zufälligen Folge zu erhalten, muß dem `Random`-Objekt ein Anfangswert gegeben werden. Dieser Anfangswert wird bei der Erzeugung des Objektes verwendet.

In Listing 4.1 finden Sie ein Java-Programm, das `Random`-Objekte auf zwei verschiedene Arten erzeugt.

Listing 4.1: Der gesamte Quelltext von `RandomNumbers.java`

```
1: import java.util.Random;
2:
3: class RandomNumbers {
4:
5:     public static void main(String arguments[]) {
6:         Random r1, r2;
7:
8:         r1 = new Random();
9:         System.out.println("Random value 1: " + r1.nextDouble());
10:
11:         r2 = new Random(8675309);
12:         System.out.println("Random value 2: " + r2.nextDouble());
13:     }
14: }
```

Wenn Sie dieses Programm kompilieren und ausführen, sollte folgendes ausgegeben werden:

```
Random value 1: 0.3125961341023068
Random value 2: 0.754788115099576
```

In diesem Beispiel werden anhand unterschiedlicher Argumente für `new` zwei verschiedene `Random`-Objekte erstellt. Die erste Instanz (Zeile 8) benutzt `new Random()` ohne Argumente, was ein `Random`-Objekt mit der aktuellen Zeit als Anfangswert erzeugt. Der Wert in der ersten Zeile Ihrer Ausgabe hängt von dem Zeitpunkt ab, zu dem Sie das Programm ausführen, da der Zufallswert das Fortschreiten der Zeit widerspiegelt. Aus diesem Grund werden die meisten `Random`-Objekte mit dem Standard-Verfahren, sprich: der Zeit als Anfangswert, erzeugt.

Der Aufruf der Methode `nextDouble()` des `Random`-Objekts in den Zeilen 9 bis 12 gibt die nächste Zahl in der

pseudozufälligen Zahlenfolge aus.

Bei der Erstellung des Random-Objekts in diesem Beispiel (Zeile 11) wird ein Integer als Argument übergeben. Die zweite Zeile der Ausgabe sollte jedesmal, wenn das Programm ausgeführt wird, denselben Wert (0.754788115099576) anzeigen. Wenn Sie ein Literal als Anfangswert übergeben, dann ist die Folge der Zufallszahlen immer dieselbe. Dies kann für Testzwecke hilfreich sein.



Sie haben vielleicht Probleme damit zu erkennen, wie man mit einer langen Dezimalzahl (z.B. 0.754788115099576) eine Zufallszahl erzeugen kann. Wenn Sie diesen Zufallswert mit einem Integer multiplizieren, dann ist das Produkt eine Zufallszahl zwischen Null und dem Integer selbst. Die folgende Anweisung multipliziert eine Zufallszahl z.B. mit 12 und speichert das Produkt als Integer:

```
Random r1 = new Random();
int number = (int)(r1.nextDouble() * 12);
```

Die Zahl wird eine Zufallszahl zwischen 0 und 12 sein.

Was new bewirkt

Bei Verwendung des new-Operators passieren mehrere Dinge: Erstens wird die neue Instanz der jeweiligen Klasse angelegt und Speicher dafür allokiert. Zweitens wird eine bestimmte Methode, die in der jeweiligen Klasse definiert ist, aufgerufen. Diese spezielle Methode nennt man *Konstruktor*.



Ein Konstruktor ist eine spezielle Methode zum Erstellen und Initialisieren neuer Instanzen von Klassen. Konstruktoren initialisieren das neue Objekt und seine Variablen, erzeugen andere Objekte, die dieses Objekt braucht, und führen im allgemeinen andere Operationen aus, die für die Ausführung des Objekts nötig sind.

Sie können in einer Klasse mehrere Konstruktor-Definitionen verwenden. Diese können sich jeweils in der Zahl und dem Typ der Argumente unterscheiden. Beim Aufruf eines Konstruktors bei der Verwendung von new wird dann anhand der übergebenen Argumente der richtige Konstruktor für jene Argumente verwendet. Auf diese Weise war es der Random-Klasse möglich, mit den verschiedenen Versionen von new unterschiedliche Aufgaben zu erfüllen. Wenn Sie eigene Klassen anlegen, können Sie beliebig viele Konstruktoren definieren, um das Verhalten einer Klasse zu bestimmen.

Speichermanagement

Wenn Sie mit anderen objektorientierten Programmiersprachen vertraut sind, werden Sie sich eventuell fragen, ob es zu der new-Anweisung ein Gegenstück gibt, das ein Objekt zerstört, sobald es nicht mehr benötigt wird.

Speichermanagement ist in Java dynamisch und automatisch. Wenn Sie in Java ein neues Objekt erstellen, wird ihm automatisch die richtige Speichergröße zugeteilt. Sie brauchen für Objekte nicht explizit Speicherplatz zuzuteilen. Das übernimmt Java für Sie.

Wenn Sie ein Objekt nicht mehr benötigen, wird der diesem Objekt zugeteilte Speicher ebenfalls automatisch wieder freigegeben. Ein Objekt, das nicht mehr gebraucht wird, hat keine aktiven Referenzen mehr (es ist keinen Variablen mehr zugewiesen, die Sie noch verwenden oder die in Arrays gespeichert sind). Java hat einen Papierkorb (garbage collector), der nach unbenutzten Objekten sucht und den diesen Objekten zugeteilten Speicherplatz zurückfordert. Sie brauchen also Speicherplatz nicht explizit freizustellen. Sie müssen nur sicherstellen, daß keine Objekte, die Sie loswerden wollen, irgendwo verwendet werden.

Verwenden von Klassen- und Instanzvariablen

Nun haben Sie ein eigenes Objekt mit definierten Klassen- oder Instanzvariablen. Wie funktionieren diese Variablen? Ganz einfach! Klassen- und Instanzvariablen verhalten sich genauso wie die lokale Variablen, die Sie gestern gelernt haben. Lediglich die Bezugnahme auf sie unterscheidet sich geringfügig von den üblichen Variablen im Code.

Werte auslesen

Um den Wert einer Instanzvariablen auszulesen, verwenden Sie die *Punkt-Notation*.

Bei der Punkt-Notation hat die Referenz auf eine Instanz- oder Klassenvariable zwei Teile: das Objekt auf der linken Seite des Punkts und die Variable rechts davon.



Die *Punkt-Notation* ist eine Methode, um auf Instanzvariablen und Methoden innerhalb eines Objekts mit einem Punkt-Operator (».«) zuzugreifen.

Ist beispielsweise ein Objekt der Variablen `myCustomer` zugewiesen und hat dieses Objekt eine Variable namens `orderTotal`, nehmen Sie auf den Wert dieser Variablen wie folgt Bezug:

```
myCustomer.orderTotal;
```

Diese Art des Zugreifens auf Variablen ist ein Ausdruck (der einen Wert ausgibt), und was auf beiden Seiten des Punkts steht, ist ebenfalls ein Ausdruck. Das bedeutet, daß Sie den Zugriff auf Instanzvariablen verschachteln können. Beinhaltet die `orderTotal`- Instanzvariable selbst ein Objekt und dieses Objekt eine eigene Instanzvariable namens `layaway`, können Sie wie folgt darauf Bezug nehmen:

```
myCustomer.orderTotal.layaway;
```

Punktausdrücke werden von links nach rechts ausgewertet, deshalb beginnen sie mit der Variablen `orderTotal` von `myCustomer`, das auf ein anderes Objekt verweist, das die Variable `layaway` enthält. Letztendlich erhalten Sie den Wert der `layaway`-Variablen.

Werte ändern

Die Zuweisung eines Wertes zu dieser Variablen ist ebenso einfach. Sie setzen einfach einen Zuweisungsoperator rechts neben den Ausdruck:

```
myCustomer.orderTotal.layaway = true;
```

Dieses Beispiel setzt den Wert der Variablen `layaway` auf `true`.

Listing 4.2 ist ein Beispiel eines Programms, das die Instanzvariablen in einem `Point`- Objekt testet und ändert. `Point` ist Teil des Pakets `java.awt` und bezieht sich auf einen Koordinatenpunkt mit einem `x`- und einem `y`-Wert.

Listing 4.2: Der gesamte Quelltext von `SetPoints.java`

```
1: import java.awt.Point;
2:
3: class SetPoints {
4:
5:     public static void main(String arguments[]) {
6:         Point location = new Point(4, 13);
7:     }
```



```

8:      System.out.println("Starting location:");
9:      System.out.println("X equals " + location.x);
10:     System.out.println("Y equals " + location.y);
11:
12:     System.out.println("\nMoving to (7, 6)");
13:     location.x = 7;
14:     location.y = 6;
15:
16:     System.out.println("\nEnding location:");
17:     System.out.println("X equals " + location.x);
18:     System.out.println("Y equals " + location.y);
19:     }
20: }

```

Wenn Sie diese Applikation ausführen, sollten Sie die folgende Ausgabe erhalten:

```

Starting location:
X equals 4
Y equals 13

```

```

Moving to (7, 6)

```

```

Ending location:
X equals 7
Y equals 6

```

In diesem Beispiel erstellen Sie zuerst eine Instanz von Point, wobei X gleich 4 und Y gleich 13 ist (Zeile 6). Die Zeilen 9 und 10 geben diese Einzelwerte über die Punkt-Notation aus. Die Zeilen 13 und 14 ändern den Wert von x auf 7 bzw. den Wert von y auf 6. Die Zeilen 17 und 18 geben die Werte von X und Y in der geänderten Form wieder aus.

Klassenvariablen

Wie Sie bereits gelernt haben, werden Klassenvariablen in der Klasse selbst definiert und gespeichert. Deshalb wirken sich ihre Werte auf die Klasse und alle ihre Instanzen aus.

Bei Instanzvariablen erhält jede neue Instanz der Klasse eine neue Kopie der Instanzvariablen, die diese Klasse definiert. Jede Instanz kann dann die Werte dieser Instanzvariablen ändern, ohne daß sich das auf andere Instanzen auswirkt. Bei Klassenvariablen gibt es nur eine Kopie der Variablen. Jede Instanz der Klasse hat Zugang zu der Variablen, jedoch gibt es nur einen Wert. Durch Änderung des Wertes dieser Variablen ändern sich die Werte aller Instanzen der betreffenden Klasse.

Sie deklarieren Klassenvariablen, indem Sie das Schlüsselwort `static` vor die Variable setzen. Betrachten wir als Beispiel folgende teilweise Klassendefinition:

```

class FamilyMember {
    static String surname = "Igwebuike";
    String name;
    int age;
}

```

Instanzen der Klasse `FamilyMember` haben je einen eigenen Wert für Name (`name`) und Alter (`age`). Die Klassenvariable Nachname (`surname`) hat aber nur einen Wert für alle Familienmitglieder: »Igwebuike«. Ändern Sie `surname`, wirkt sich das auf alle Instanzen von `FamilyMember` aus.



Die Bezeichnung *statisch* (über das Schlüsselwort *static*) für diese Variablen bezieht sich auf eine Bedeutung des Wortes: ortsfest. Wenn eine Klasse eine statische Variable besitzt, dann hat diese Variable in jedem Objekt dieser Klasse denselben Wert.

Um auf Klassenvariablen zuzugreifen, benutzen Sie die gleiche Punkt-Notation wie bei Instanzvariablen. Um den Wert der Klassenvariablen auszulesen oder zu ändern, können Sie entweder die Instanz oder den Namen der Klasse links neben dem Punkt eingeben. Beide Ausgabezeilen in diesem Beispiel geben den gleichen Wert aus:

```
FamilyMember dad = new FamilyMember();
System.out.println("Family's surname is: " + dad.surname);
System.out.println("Family's surname is: " + FamilyMember.surname);
```

Da Sie eine Instanz benutzen können, um den Wert einer Klassenvariablen zu ändern, entsteht leicht Verwirrung über Klassenvariablen und darüber, wo der Wert herkommt (Sie erinnern sich, daß sich der Wert einer Klassenvariablen auf alle Instanzen auswirkt). Aus diesem Grund empfiehlt es sich, den Namen der Klasse zu verwenden, wenn auf eine Klassenvariable verwiesen wird. Dadurch wird der Code besser lesbar, und Fehler lassen sich schneller finden.

Aufrufen von Methoden

Das Aufrufen von Methoden in Objekten läuft ähnlich ab wie die Bezugnahme auf seine Instanzvariablen: Auch in Methodenaufrufen wird die Punkt-Notation benutzt. Das Objekt, dessen Methode Sie aufrufen, steht links neben dem Punkt. Der Name der Methode und ihre Argumente stehen rechts neben dem Punkt:

```
myCustomer.addToOrder(itemNumber, price, quantity);
```

Beachten Sie, daß nach jeder Methode Klammern folgen müssen, auch wenn die Methode keine Argumente hat:

```
myCustomer.cancelAllOrders();
```

Gibt die aufgerufene Methode ein Objekt zurück, das selbst Methoden hat, können Sie Methoden wie Variablen verschachteln. Das nächste Beispiel ruft die Methode `talkToManager()` auf. Diese ist in dem Objekt definiert, das von der Methode `cancelAllOrders()` zurückgegeben wird, die wiederum in dem Objekt `myCustomer` definiert ist:

```
myCustomer.cancelAllOrders().talkToManager();
```

Sie können auch verschachtelte Methodenaufrufe und Referenzen auf Instanzvariablen kombinieren. Im nächsten Beispiel wird die Methode `putOnLayaway()` aufgerufen. Diese ist in dem Objekt definiert, das in der Instanzvariablen `orderTotal` gespeichert ist. Die Instanzvariable selbst ist Teil des `myCustomer`-Objekts:

```
myCustomer.orderTotal.putOnLayaway(itemNumber, price, quantity);
```

`System.out.println()`, die Methode, die Sie bisher in diesem Buch benutzt haben, um Text auszugeben, ist ein gutes Beispiel für die Verschachtelung von Variablen und Methoden. Die `System`-Klasse (Teil des Pakets `java.lang`) beschreibt systemspezifisches Verhalten. `System.out` ist eine Klassenvariable, die eine Instanz der Klasse `PrintStream` enthält. Diese Instanz zeigt auf die Standardausgabe des Systems. `PrintStream`-Instanzen enthalten die Methode `println()`, die eine Zeichenkette an diesen Ausgabestream schickt.

Listing 4.3 zeigt ein Beispiel des Aufrufens einiger Methoden, die in der `String`-Klasse definiert sind. `String`-Objekte beinhalten Methoden zum Testen und Ändern von Strings auf ähnliche Weise, wie man sie von einer Stringbibliothek in anderen Sprachen erwarten würde.

Listing 4.3: Der gesamte Quelltext von `CheckString.java`

```
1: class CheckString {
2:
3:     public static void main(String arguments[]) {
4:         String str = "In my next life, I will believe in reincarnation";
```

```

5:      System.out.println("The string is: " + str);
6:      System.out.println("Length of this string: "
7:          + str.length());
8:      System.out.println("The character at position 7: "
9:          + str.charAt(7));
10:     System.out.println("The substring from 24 to 31: "
11:         + str.substring(24, 31));
12:     System.out.println("The index of the character x: "
13:         + str.indexOf('x'));
14:     System.out.println("The index of the beginning of the "
15:         + "substring \"will\": " + str.indexOf("will"));
16:     System.out.println("The string in upper case: "
17:         + str.toUpperCase());
18: }
19: }

```

Folgendes gibt das Programm auf dem Standardausgabegerät aus:

```

The string is: In my next life, I will believe in reincarnation
Length of this string: 48
The character at position 7: e
The substring from 24 to 31: believe
The index of the character x: 8
The index of the beginning of the substring "will": 19
The string in upper case: IN MY NEXT LIFE, I WILL BELIEVE IN REINCARNATION

```

In Zeile 4 erstellen Sie eine neue Instanz von String durch Verwendung eines Zeichenkettenliterals (das ist einfacher, als wenn man `new` verwendet und die Zeichen dann einzeln eingibt). Der Rest des Programms ruft verschiedene String-Methoden auf, um verschiedene Operationen auf diese Zeichenkette auszuführen:

- Zeile 5 gibt den Wert der in Zeile 4 geschriebenen Zeichenkette aus: »In my next life, I will believe in reincarnation.«
- Zeile 7 ruft die Methode `length()` im neuen String-Objekt auf. Diese Zeichenkette hat 48 Zeichen.
- Zeile 9 ruft die Methode `charAt()` auf, die das Zeichen an der angegebenen Position ausgibt. Beachten Sie, daß Zeichenketten mit der Position 0 beginnen, deshalb ist das Zeichen an Position 7 ein e.
- Zeile 11 ruft die Methode `substring()` auf, die zwei Ganzzahlen benutzt, um einen Bereich festzulegen, und die Teilzeichenkette zwischen diesen Anfangs- und Endpunkten ausgibt. Die Methode `substring()` kann auch mit nur einem Argument aufgerufen werden. Dadurch wird die Teilzeichenkette von dieser Position bis zum Ende der Zeichenkette ausgegeben.
- Zeile 13 ruft die Methode `indexOf()` auf, die die Position der ersten Instanz eines bestimmten Zeichens (hier 'x') ausgibt. Zeichenliterals werden im Gegensatz zu Stringliterals in einfachen Anführungszeichen eingeschlossen. Wäre das 'x' in Zeile 13 in doppelte Anführungszeichen eingeschlossen, dann würde es wie ein String-Objekt behandelt werden.
- Zeile 15 zeigt eine andere Verwendung der Methode `indexOf()`, die hier ein String-Argument verwendet und den Index des Beginns dieser Zeichenkette ausgibt.
- Zeile 17 benutzt die Methode `toUpperCase()`, die eine Kopie des Strings in Großbuchstaben ausgibt.

Klassenmethoden

Klassenmethoden wirken sich wie Klassenvariablen auf die ganze Klasse, nicht auf ihre einzelnen Instanzen, aus. Klassenmethoden werden üblicherweise für allgemeine Utility-Methoden benutzt, die nicht direkt auf eine Instanz der Klasse ausgeführt werden sollen, sondern lediglich vom Konzept her in diese Klasse passen. Die String-Klasse enthält beispielsweise die Klassenmethode `valueOf()`, die einen von vielen verschiedenen Argumenttypen (Ganzzahlen, boolesche Operatoren, andere Objekte usw.) verarbeiten kann. Die Methode

`valueOf()` gibt dann eine neue Instanz von `String` aus, die den Zeichenkettenwert des Arguments enthält. Diese Methode wird nicht direkt auf eine vorhandene Instanz von `String` ausgeführt. Das Konvertieren eines Objekts oder Datentyps in einen `String` ist definitiv eine Operation, die in die `String`-Klasse paßt. Deshalb ist es sinnvoll, sie gleich in der `String`-Klasse zu definieren.

Klassenmethoden sind auch nützlich, um allgemeine Methoden an einer Stelle (der Klasse) zusammenzufassen. Die `Math`-Klasse, die im Paket `java.lang` enthalten ist, umfaßt beispielsweise zahlreiche mathematische Operationen als Klassenmethoden. Es gibt keine Instanzen der Klasse `Math`. Trotzdem können Sie ihre Methoden mit numerischen oder booleschen Argumenten verwenden. Die Klassenmethode `Math.max()` erwartet z.B. zwei Argumente und gibt das größere der beiden zurück. Sie müssen dafür keine neue Instanz der Klasse `Math` erzeugen. Sie können diese Methode immer dort aufrufen, wo Sie sie gerade benötigen, wie das im folgenden der Fall ist:

```
int maximumPrice = Math.max(firstPrice, secondPrice);
```

Um eine Klassenmethode aufzurufen, benutzen Sie die Punkt-Notation wie bei Instanzmethoden. Ebenso wie bei Klassenvariablen können Sie entweder eine Instanz der Klasse oder die Klasse selbst links neben den Punkt setzen. Allerdings ist die Verwendung des Namens der Klasse für Klassenvariablen aus den gleichen Gründen, die im Zusammenhang mit Klassenvariablen erwähnt wurden, empfehlenswert, da der Code dadurch übersichtlicher wird. Die letzten zwei Zeilen dieses Beispiels produzieren das gleiche Ergebnis - den `String` »5«:

```
String s, s2;
s = "foo";
s2 = s.valueOf(5);
s2 = String.valueOf(5);
```

Referenzen auf Objekte

Wie bei der Arbeit mit Objekten ist die Verwendung von Referenzen, die auf diese Objekte zeigen, ein wichtiger Aspekt.



Eine *Referenz* ist eine Art Zeiger (Pointer), der auf ein Objekt verweist.

Wenn Sie Objekte Variablen zuweisen oder Objekte als Argumente an Methoden weiterreichen, legen Sie Referenzen auf diese Objekte fest. Die Objekte selbst oder Kopien davon werden dabei nicht weitergereicht.

Ein Beispiel soll dies verdeutlichen. Sehen Sie sich den Code in Listing 4.4 an.

Listing 4.4: Der gesamte Quelltext von `ReferencesTest.java`

```
1: import java.awt.Point;
2:
3: class ReferencesTest {
4:     public static void main (String arguments[]) {
5:         Point pt1, pt2;
6:         pt1 = new Point(100, 100);
7:         pt2 = pt1;
8:
9:         pt1.x = 200;
10:        pt1.y = 200;
11:        System.out.println("Point1: " + pt1.x + ", " + pt1.y);
12:        System.out.println("Point2: " + pt2.x + ", " + pt2.y);
13:    }
14: }
```

Das folgende stellt die Ausgabe des Programms dar:

Point1: 200, 200

Point2: 200, 200

Folgendes passiert im ersten Teil des Programms:

- Zeile 5: Es werden zwei Variablen vom Typ Point erstellt.
- Zeile 6: Ein neues Point-Objekt wird pt1 zugewiesen.
- Zeile 7: Der Wert von pt1 wird pt2 zugewiesen.

Die Zeilen 9 bis 12 sind der trickreiche Teil. Die Variablen x und y von pt1 werden beide auf 200 gesetzt. Anschließend werden alle Variablen von pt1 und pt2 auf den Bildschirm ausgegeben.

Sie erwarten eventuell, daß pt1 und pt2 unterschiedliche Werte haben. Die Ausgabe zeigt allerdings, daß dies nicht der Fall ist. Wie Sie sehen können, wurden die Variablen von pt2 auch geändert, obwohl nichts unternommen wurde, um diese zu ändern.

Die Ursache dafür ist, daß in Zeile 7 eine Referenz auf das Objekt in pt1 in pt2 erzeugt wird, anstatt pt2 als neues Objekt zu erstellen und pt1 in dieses zu kopieren.

pt2 ist eine Referenz auf dasselbe Objekt, das sich auch in pt1 befindet. Abbildung 4.1 verdeutlicht dies. Jede der beiden Variablen kann dazu verwendet werden, auf das Objekt zuzugreifen oder dessen Variablen zu verändern.

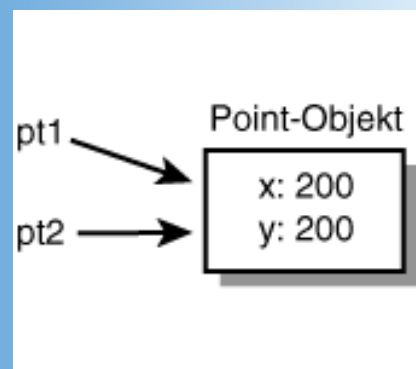


Abbildung 4.1:
Referenzen auf ein Objekt

Wenn Sie wollen, daß pt1 und pt2 sich auf verschiedene Objekte beziehen, verwenden Sie in den Zeilen 6 und 7 `new Point()`-Anweisungen, wie im folgenden, um diese zu erzeugen:

```
pt1 = new Point(100, 100);
pt2 = new Point(100, 100);
```

Die Tatsache, daß Java Referenzen benutzt, gewinnt besondere Bedeutung, wenn Sie Argumente an Methoden weiterreichen. Sie lernen hierüber noch in dieser Lektion mehr.



In Java gibt es keine explizite Zeigerarithmetik oder Zeiger (*Pointer*), sondern nur Referenzen. Mit Java-Referenzen haben Sie aber die gleichen Möglichkeiten zur Hand wie mit Zeigern, allerdings ohne deren Nachteile.

Casting und Konvertieren von Objekten und Primitivtypen

Etwas werden Sie über Java sehr schnell herausfinden: Java ist sehr pingelig in bezug auf die Informationen, die es verarbeitet. Java erwartet, daß die Informationen eine bestimmte Form haben, und läßt Alternativen nicht zu. Wenn Sie Argumente an Methoden übergeben oder Variablen in Ausdrücken verwenden, dann müssen Sie Variablen mit den richtigen Datentypen verwenden. Wenn eine Methode einen `int` erwartet, wird der Java-Compiler mit einem Fehler reagieren, falls Sie versuchen einen `float`-Wert an die Methode zu übergeben. Entsprechendes gilt, wenn Sie einer Variablen den Wert einer anderen zuweisen - beide müssen desselben Typs sein.

Es gibt einen Bereich, in dem der Java-Compiler nicht so pingelig ist: Strings. Die Verarbeitung von Strings in `println()`-Methoden, Zuweisungen und Methodenargumenten ist durch den Verkettungsoperator (`»+«`) stark vereinfacht worden. Wenn eine Variable in einer Gruppe von verketteten Variablen ein String ist, dann behandelt Java das Ganze als String. Dadurch wird folgendes möglich:

```
float gpa = 2.25F;
System.out.println("Honest, dad, my GPA is a " + (gpa+1.5));
```

Manchmal werden Sie in einem Java-Programm einen Wert haben, der nicht den richtigen Typ für das, was Sie damit tun wollen, hat. Er weist vielleicht die falsche Klasse oder den falschen Datentyp auf - z.B. `float`, wenn Sie `int` benötigen.

Um einen Wert von einem Typ in einen anderen zu konvertieren, verwenden Sie das sogenannte Casting.



Casting ist ein Mechanismus, um einen neuen Wert zu erstellen, der einen anderen Typ aufweist als dessen Quelle. Casting ergibt ein neues Objekt oder einen neuen Wert. Casting wirkt sich nicht auf das ursprüngliche Objekt bzw. den ursprünglichen Wert aus.

Obwohl das Casting-Konzept an sich einfach ist, werden die Regeln, die bestimmen, welche Typen in Java in andere konvertiert werden können, durch die Tatsache komplizierter, daß Java sowohl primitive Typen (`int`, `float`, `boolean`) als auch Objekttypen (`String`, `Point`, `Window` usw.) hat. Aufgrund dieser drei Typen gibt es drei Formen von Casting und Umwandlungen, über die wir in dieser Lektion sprechen:

- Casting von primitiven Typen: `int` in `float` in `boolean`.
- Casting zwischen Objekttypen: eine Klasseninstanz wird in eine Instanz einer anderen Klasse konvertiert.
- Konvertieren von primitiven Typen in Objekte und Herausziehen der primitiven Werte, um sie den Objekten zurückzugeben.

Es ist vielleicht leichter, bei der folgenden Besprechung des Casting von Quellen und Zielen auszugehen. Die Quelle ist die Variable, die in einen anderen Typ gecastet wird. Das Ziel ist das Ergebnis.

Konvertieren von Primitivtypen

Durch Konvertieren zwischen primitiven Typen können Sie den Wert eines Typs in einen anderen primitiven Typ umwandeln, z.B. um eine Zahl eines Typs einer Variablen zuzuweisen, die auf einem anderen Typ basiert. Das Casting tritt bei primitiven Typen am häufigsten bei numerischen Typen auf. Boolesche Werte können nicht in einen anderen Primitivtyp konvertiert werden. Sie können aber 1 oder 0 in boolesche Werte konvertieren.

In vielen Casts zwischen primitiven Typen kann das Ziel größere Werte als die Quelle aufnehmen, so daß der Wert ohne Schwierigkeiten konvertiert werden kann. Ein Beispiel hierfür wäre die Konvertierung eines `byte` in einen `int`. Da ein `byte` nur Werte von -128 bis 127 aufnehmen kann und ein `int` Werte von -2.1 Millionen bis 2.1

Millionen, ist mehr als genug Platz für den Wert in einem byte.

Meist kann ein byte oder ein char automatisch als int oder ein int als long, ein int als float oder etwas anderes als double behandelt werden. In diesem Fall gehen beim Konvertieren des Wertes keine Informationen verloren, weil der größere Typ mehr Genauigkeit bietet als der kleinere.



Ein Zeichen (char) kann als int verwendet werden, da jedes Zeichen einen korrespondierenden numerischen Wert hat, der die Position des Zeichens innerhalb des Zeichensatzes angibt. Wenn die Variable i den Wert 65 hat, liefert der Cast (char)i das Zeichen 'A'. Der numerische Code für A ist nach dem ASCII-Zeichensatz 65. Der ASCII-Zeichensatz ist Teil der Zeichenunterstützung von Java.

Um einen großen Wert auf einen kleineren Typ zu konvertieren, müssen Sie ein explizites Casting anwenden, weil bei dieser Umsetzung der Wert an Genauigkeit einbüßen kann. Explizites Casting sieht so aus:

(Typname) Wert

In dieser Form ist Typname der Name des Typs, auf den Sie konvertieren (z.B. short, int, float, boolean), und Wert ist ein Ausdruck, der den zu konvertierenden Wert ergibt. Dieser Ausdruck teilt den Wert von x durch den Wert von y und wandelt das Ergebnis in int um:

(int) (x / y);

Da Casting eine höhere Präzedenz hat als Arithmetik, müssen Sie Klammern eingeben, damit das Ergebnis der Division an das konvertierte int übergibt. Ansonsten würde als erstes der Wert von x in einen int gecastet und dieser würde anschließend durch y geteilt werden, was natürlich einen anderen Wert ergeben könnte.

Konvertieren von Objekten

Mit einer Einschränkung können auch Klasseninstanzen in Instanzen anderer Klassen konvertiert werden: Die Quell- und die Zielklasse müssen durch Vererbung miteinander verbunden sein. Eine Klasse muß die Subklasse einer anderen sein.

Wie beim Konvertieren eines primitiven Wertes in einen größeren Typ müssen bestimmte Objekte nicht unbedingt explizit konvertiert werden. Insbesondere, weil die Subklassen von Instanzen normalerweise alle Informationen ihrer Superklassen enthalten, können Sie eine Instanz einer Subklasse überall dort verwenden, wo eine Superklasse erwartet wird.

Nehmen wir z.B. an, Sie haben eine Methode mit zwei Argumenten: eines vom Typ Object und eines vom Typ Window. Sie müssen nun nicht Instanzen dieser beiden Klassen an die Methode übergeben. Für das Object-Argument können Sie jede beliebige Subklasse von Object (anders ausgedrückt, jedes Objekt, da in Java alle Klassen Subklassen der Klasse Object sind) und für das Window-Argument jede Instanz einer beliebigen Subklasse von Window (Dialog, FileDialog und Frame) weitergeben.

Dies gilt an beliebiger Stelle in einem Programm - nicht nur in Methodenaufrufen. Wenn Sie eine Variable vom Typ Window deklariert haben, können Sie ihr Objekte dieser oder einer ihrer Subklassen zuweisen, ohne ein Casting ausführen zu müssen.

Dies gilt auch in der umgekehrten Richtung. Sie können eine Superklasse angeben, wenn eine Subklasse erwartet wird. Da allerdings Subklassen mehr Information als deren Superklassen enthalten, ist dies mit einem Verlust an Genauigkeit verbunden. Die Objekte der Superklassen haben eventuell nicht alle Verhaltensweisen und Eigenschaften, um anstelle eines Objekts der Subklasse zu arbeiten. Wenn Sie z.B. eine Operation verwenden, die Methoden in einem Objekt der Klasse Integer aufruft, kann es sein, daß ein Objekt der Klasse Number diese Methoden nicht beinhaltet, da diese in Integer definiert sind. Es treten Fehler auf, wenn Sie versuchen, Methoden aufzurufen, die das Zielobjekt nicht unterstützt.

Um Objekte einer Superklasse dort zu verwenden, wo eigentlich Objekte von Subklassen erwartet werden, müssen Sie diese explizit casten. Sie werden keine Informationen bei dieser Konvertierung verlieren. Statt dessen erhalten Sie alle Methoden und Variablen, die die Subklasse definiert. Um ein Objekt in eine andere Klasse zu casten, verwenden Sie dieselbe Operation, die Sie auch für primitive Typen verwenden:

(Klassenname) Objekt

In diesem Fall ist Klassenname der Name der Klasse, in die Sie das Objekt konvertieren wollen, und Objekt ist eine Referenz auf das konvertierte Objekt. Das Casting erstellt eine neue Instanz der neuen Klasse mit allen Informationen, die das alte Objekt enthielt. Das alte Objekt besteht unverändert fort.

Nachfolgend ein fiktives Beispiel, in dem eine Instanz der Klasse VicePresident in eine Instanz der Klasse Employee konvertiert wird, wobei VicePresident eine Subklasse von Employee ist, die weitere Informationen definiert (z.B. daß der Vice-President besondere Privilegien in den Waschräumen hat):

```
Employee emp = new Employee();
VicePresident veep = new VicePresident();
emp = veep; // kein Casting in dieser Richtung nötig
veep = (VicePresident)emp; // muß explizit gecastet werden
```

Casting ist auch immer dann nötig, wenn Sie die neuen 2D-Zeichenoperationen verwenden, die mit Java 1.2 eingeführt wurden. Sie müssen ein Graphics-Objekt in ein Graphics2D-Objekt casten, bevor Sie auf den Bildschirm Grafikausgaben tätigen können. Das folgende Beispiel verwendet ein Graphics-Objekt namens screen, um ein neues Graphics2D-Objekt zu erzeugen, das den Namen screen2D trägt:

```
Graphics2D screen2D = (Graphics2D)screen;
```

Graphics2D ist eine Subklasse von Graphics, und beide befinden sich im Paket java.awt . Sie werden das Thema ausführlich am Tag 9 kennenlernen.

Abgesehen vom Konvertieren von Objekten in Klassen können Sie auch Objekte in Schnittstellen konvertieren, jedoch nur, wenn die Klasse oder eine Superklasse des Objekts die Schnittstelle implementiert. Durch Casting eines Objekts in eine Schnittstelle können Sie dann eine der Methoden dieser Schnittstelle aufrufen, auch wenn die Klasse des Objekts diese Schnittstelle nicht direkt implementiert.

Konvertieren von Primitivtypen in Objekte und umgekehrt

Etwas, was unter keinen Umständen möglich ist, ist die Konvertierung eines Objekts in einen primitiven Datentyp oder umgekehrt. Primitive Datentypen und Objekte sind in Java völlig verschiedene Dinge, und es ist nicht möglich, zwischen diesen automatisch zu konvertieren oder sie im Austausch zu verwenden.

Als Alternative enthält das Paket java.lang mehrere Sonderklassen, die je einem primitiven Datentyp entsprechen: Integer, Float, Boolean usw. Beachten Sie bitte, daß die Namen dieser Klassen mit einem Großbuchstaben und die Namen der primitiven Datentypen mit einem Kleinbuchstaben beginnen. Java behandelt die Datentypen und deren Klassenversionen sehr unterschiedlich, und ein Programm kann nicht erfolgreich kompiliert werden, wenn Sie die eine Variante verwenden und die andere erwartet wird. Mit den in diesen Klassen definierten Klassenmethoden können Sie anhand von new für alle primitive Typen ein Gegenstück in Form eines Objekts erstellen. Die folgenden Codezeilen erstellen eine Instanz der Klasse Integer mit dem Wert 4403:

```
Integer intObject = new Integer(4403);
```

Sobald Sie auf diese Art ein Objekt erzeugt haben, können Sie es wie jedes andere Objekt verwenden. Möchten Sie die primitiven Werte zurückkonvertieren, gibt es auch dafür Methoden. Wenn Sie z.B. einen int-Wert aus einem dataCount-Objekt herausziehen wollen, könnten Sie die folgende Anweisung verwenden:

```
int theInt = intObject.intValue(); // gibt 4403 aus
```

In Programmen werden Sie sehr häufig die Konvertierung von String-Objekten in numerische Typen, wie Integer, benötigen. Wenn Sie einen int als Ergebnis benötigen, dann können Sie dafür die Methode parseInt() der Klasse Integer verwenden. Der String, der konvertiert werden soll, ist das einzige Argument, das dieser Methode

übergeben wird. Das folgende Beispiel zeigt dies:

```
String pennsylvania = "65000";
int penn = Integer.parseInt(pennsylvania);
```

Schlagen Sie in der *Java-API-Dokumentation* über diese speziellen Klassen nach. Sie finden dort Erklärungen der Methoden zum Konvertieren von Primitivtypen in Objekte und umgekehrt. Die Dokumentation können Sie auf der Website von JavaSoft (<http://java.sun.com>) online lesen oder sich herunterladen.



Es gibt spezielle Typklassen für Boolean, Byte, Character, Double, Float, Integer, Long, Short und Void.

Objekte vergleichen und mehr

Neben dem Casting gibt es noch weitere Operationen, die Sie auf Objekte anwenden können:

- Vergleichen von Objekten
- Ermitteln der Klasse eines bestimmten Objekts
- Ermitteln, ob ein Objekt eine Instanz einer bestimmten Klasse ist

Vergleichen von Objekten

Gestern haben Sie Operatoren zum Vergleichen von Werten kennengelernt: gleich, ungleich, kleiner als usw. Die meisten dieser Operatoren funktionieren nur mit primitiven Typen, nicht mit Objekten. Falls Sie versuchen, andere Werte als Operanden zu verwenden, gibt der Java-Compiler Fehler aus.

Die Ausnahme zu dieser Regel bilden die Operatoren für Gleichheit: == (gleich) und != (ungleich). Wenn Sie diese Operatoren auf Objekte anwenden, hat dies nicht den Effekt, den Sie zunächst erwarten werden. Anstatt zu prüfen, ob ein Objekt denselben Wert wie ein anderes Objekt hat, prüfen diese Operatoren, ob es sich bei den beiden Objekten um dasselbe Objekt handelt.

Um Instanzen Ihrer Klasse zu vergleichen und aussagefähige Ergebnisse zu erzielen, müssen Sie spezielle Methoden in Ihre Klasse implementieren und diese Methoden aufrufen.

Ein gutes Beispiel dafür ist die String-Klasse. Es ist möglich, daß zwei String-Objekte dieselben Werte beinhalten. Nach dem Operator == sind diese zwei String-Objekte aber nicht gleich, weil sie zwar den gleichen Inhalt haben, aber nicht dasselbe Objekt sind.

Um festzustellen, ob zwei String-Objekte den gleichen Inhalt haben, definiert die String-Klasse die Methode equals(), die jedes Zeichen in der Zeichenkette prüft und true ausgibt, wenn die zwei Zeichenketten die gleichen Werte haben. Dies wird in Listing 4.5 verdeutlicht.

Listing 4.5: Der komplette Quelltext von EqualsTest.java

```
1: class EqualsTest {
2:     public static void main(String args[]) {
3:         String str1, str2;
4:         str1 = "Free the bound periodicals.";
5:         str2 = str1;
6:
7:         System.out.println("String1: " + str1);
8:         System.out.println("String2: " + str2);
9:         System.out.println("Same object? " + (str1 == str2));
10:
11:         str2 = new String(str1);
```



```

12:
13:         System.out.println("String1: " + str1);
14:         System.out.println("String2: " + str2);
15:         System.out.println("Same object? " + (str1 == str2));
16:         System.out.println("Same value? " + str1.equals(str2));
17:     }
18: }

```

Das Programm erzeugt die folgende Ausgabe:

```

String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? true
String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? false
Same value? true

```

Der erste Teil dieses Programms (Zeilen 3 bis 5) deklariert die zwei Variablen `str1` und `str2`, weist das Literal "Free the bound periodicals" `str1` und dann diesen Wert `str2` zu. Wie Sie von Objektreferenzen her wissen, zeigen `str1` und `str2` jetzt auf dasselbe Objekt. Das beweist der Test in Zeile 9.

Im zweiten Teil wird ein neues String-Objekt mit dem Wert von `str1` erstellt. Jetzt bestehen zwei verschiedene String-Objekte mit dem gleichen Wert. Sie werden mit dem Operator `==` (in Zeile 15) geprüft, um zu ermitteln, ob sie das gleiche Objekt sind. Die erwartete Antwort wird ausgegeben. Schließlich erfolgt das Prüfen mit der `equals()`-Methode (in Zeile 16), es liefert auch das erwartete Ergebnis (`true` - beide haben den gleichen Wert).



Warum kann man anstelle von `new` nicht einfach ein anderes Literal verwenden, wenn man `str2` ändert? String-Literale sind in Java optimiert. Wenn Sie ein String mit einem Literal erstellen und dann ein anderes Literal mit den gleichen Zeichen benutzen, weiß Java genug, um Ihnen das erste String-Objekt zurückzugeben. Die beiden Strings sind das gleiche Objekt. Um zwei separate Objekte zu erstellen, müssten Sie sehr umständlich vorgehen.

Bestimmen der Klasse eines Objekts

Möchten Sie die Klasse eines Objekts ermitteln? Hier ist eine Möglichkeit, dies bei einem Objekt zu erreichen, das der Variablen `obj` zugewiesen ist:

```
String name = obj.getClass().getName();
```

Was geschieht hier? Die Methode `getClass()` ist in der Klasse `Object` definiert und als solche für alle Objekte verfügbar. Das Ergebnis dieser Methode ist ein `Class`-Objekt (wobei `Class` selbst eine Klasse ist), die die Methode `getName()` hat. `getName()` gibt den Namen der Klasse als Zeichenkette aus.

Einen anderen nützlichen Test bietet der Operator `instanceof`. `instanceof` hat zwei Operanden: ein Objekt links und den Namen einer Klasse rechts. Der Ausdruck gibt `true` oder `false` aus, je nachdem, ob das Objekt eine Instanz der benannten Klasse oder eines der Superklassen dieser Klasse ist:

```

"swordfish" instanceof String // true
Point pt = new Point(10, 10);
pt instanceof String // false

```

Der Operator `instanceof` kann auch für Schnittstellen benutzt werden. Falls ein Objekt eine Schnittstelle implementiert, gibt der `instanceof`-Operator mit einem Schnittstellennamen auf der rechten Seite `true` aus.

Klassen und Methoden mit Reflexion inspizieren

Eine der Verbesserungen von Java nach dem Release 1.0.2 ist die Einführung von Reflexion, auch als Introspection bezeichnet. Reflexion bzw. Introspection ermöglicht es einer Java-Klasse - beispielsweise einem von Ihnen geschriebenen Programm - Details über eine beliebige andere Klasse zu ermitteln.

Mit Reflexion kann ein Java-Programm eine Klasse laden, von der es nichts weiß, die Variablen, Methoden und Konstruktoren dieser Klasse ermitteln und damit arbeiten.

Das ergibt wahrscheinlich mehr Sinn, wenn Sie es an einem Beispiel sehen. Listing 4.6 ist eine kleine Java-Applikation namens SeeMethods.

Listing 4.6: Der komplette Text von SeeMethods.java

```

1: import java.lang.reflect.*;
2: import java.util.Random;
3:
4: class SeeMethods {
5:     public static void main(String[] arguments) {
6:         Random rd = new Random();
7:         Class className = rd.getClass();
8:         Method[] methods = className.getMethods();
9:         for (int i = 0; i < methods.length; i++) {
10:            System.out.println("Method: " + methods[i]);
11:        }
12:    }
13: }
```

In diesem Programm wird die `java.lang.reflect.*`-Klassengruppe genutzt, die Informationen über die Attribute, Methoden und Konstruktor-Methoden beliebiger Klassen liefert.

Die Applikation SeeMethods erzeugt in Zeile 6 ein Random-Objekt und nutzt dann Reflexion zur Anzeige aller öffentlichen Methoden, die ein Teil dieser Klasse sind. Listing 4.7 zeigt die Ausgabe der Applikation.

Listing 4.7: Die Ausgabe der Applikation SeeMethods

```

1: Method: public final native java.lang.Class java.lang.Object.getClass()
2: Method: public native int java.lang.Object.hashCode()
3: Method: public boolean java.lang.Object.equals(java.lang.Object)
4: Method: public java.lang.String java.lang.Object.toString()
5: Method: public final native void java.lang.Object.notify()
6: Method: public final native void java.lang.Object.notifyAll()
7: Method: public final native void java.lang.Object.wait(long) throws
java.lang.InterruptedException
8: Method: public final void java.lang.Object.wait(long,int) throws
java.lang.InterruptedException
9: Method: public final void java.lang.Object.wait() throws
java.lang.InterruptedException
10: Method: public synchronized void java.util.Random.setSeed(long)
11: Method: public void java.util.Random.nextBytes(byte[])
12: Method: public int java.util.Random.nextInt()
13: Method: public long java.util.Random.nextLong()
14: Method: public float java.util.Random.nextFloat()
15: Method: public double java.util.Random.nextDouble()
16: Method: public synchronized double java.util.Random.nextGaussian()
```

Durch Reflexion kann die Applikation SeeMethods die einzelnen Methoden der Random -Klasse und alle

Methoden, die sie von Random übergeordneten Klassen geerbt hat, kennenlernen. Jede Zeile im Listing zeigt folgende Informationen über eine Methode:

- Ob sie public ist oder nicht
- Welchen Objekt- oder Variablentyp die Methode zurückgibt
- Ob die Methode zur aktuellen Klasse oder zu einer Superklasse gehört
- Den Namen der Methode
- Die Art des Objekts und der Variablen, die bei Aufruf der Methode als Argumente übergeben werden

Die Applikation SeeMethods kann mit jeder Objektklasse ausgeführt werden - ändern Sie die Zeile 6 in SeeMethods.java, um ein anderes Objekt zu erzeugen und einen Blick in sein Inneres zu werfen.

Am häufigsten wird Reflexion von Tools wie beispielsweise Klassen-Browsern und -Debuggern eingesetzt, um mehr über die Klassen der durchsuchten oder getesteten Objektklasse zu erfahren. Außerdem wird Reflexion im Zusammenhang mit JavaBeans eingesetzt. Hier ist es bei der Erstellung größerer Anwendungen hilfreich, daß ein Objekt ein anderes Objekt abfragen kann, was es machen kann (und es dann auffordern kann, etwas zu tun). Über JavaBeans erfahren Sie noch mehr während Tag 19.

Das java.lang.reflect-Paket umfaßt folgende Klassen:

- Field, für die Verwaltung und das Herausfinden von Informationen über Klassen- und Instanzvariablen
- Method, für die Verwaltung von Klassen- und Instanzenmethoden
- Constructor, für die Verwaltung der Spezialmethoden zur Erstellung neuer Klasseninstanzen
- Array, für die Verwaltung von Arrays
- Modifier, für die Dekodierung von Modifier-Informationen über Klassen, Variablen und Methoden. (Das wird während Tag 16 weiter behandelt.)

Darüber hinaus ist eine Reihe neuer Methoden in einer Objektkasse namens Class verfügbar, die helfen, die verschiedenen Reflexions-Klassen zusammenzuhalten.

Reflexion ist ein fortgeschrittenes Feature, das Sie nicht einfach so in Ihren Programmen einsetzen. Es wird dann besonders nützlich, wenn Sie mit Objektserialisation, JavaBeans und anderen ausgeklügelten Java-Programmiertechniken arbeiten.

Zusammenfassung

Nun da Sie einen großen Schluck aus der Flasche der objektorientierten Programmierung mit Java genommen haben, sind Sie besser in der Lage zu entscheiden, wie nützlich dies für Ihre eigene Programmierung ist.

Wenn Sie zu der Sorte Mensch gehören, für die ein Glas bei der Hälfte halb leer ist, dann ist die objektorientierte Programmierung eine Abstraktionsebene, die sich zwischen Sie und das stellt, für das Sie die Programmiersprache verwenden wollen. Sie lernen in den nächsten Kapiteln mehr darüber, warum die OOP vollkommen in Java integriert ist.

Wenn Sie zu den Halb-voll-Menschen gehören, dann lohnt sich für Sie die Anwendung der objektorientierten Programmierung aufgrund der Vorteile, die Sie bietet: verbesserte Verlässlichkeit, Lesbarkeit und Pflfegbarkeit.

Heute haben Sie gelernt, wie Sie mit Objekten umgehen: sie erzeugen, deren Werte lesen und verändern und deren Methoden aufrufen. Sie haben außerdem gelernt, wie Sie Objekte einer Klasse in eine andere Klasse konvertieren bzw. wie Sie von einem Datentyp zu einer Klasse konvertieren.

Schließlich haben Sie einen ersten Blick auf die Reflexion geworfen; damit kann man ein Objekt dazu veranlassen, Details über sich selbst zu zeigen.

An diesem Punkt besitzen Sie die Fähigkeiten, um die meisten einfachen Aufgaben in Java zu bewältigen. Was nun noch fehlt sind Arrays, Bedingungen und Schleifen (die morgen behandelt werden) und wie Sie Klassen definieren und verwenden (wird an Tag 6 durchgenommen).

Fragen und Antworten

Frage:

Mir ist der Unterschied zwischen Objekten und den primitiven Datentypen, z.B. int und boolean, noch nicht ganz klar.

Antwort:

Die primitiven Typen (byte, short, int, long, float, double und char) sind die kleinsten Elemente der Sprache Java. Es sind keine Objekte, obwohl sie auf vielerlei Art wie Objekte gehandhabt werden. Sie können Variablen zugewiesen und zwischen Methoden weitergereicht werden. Die meisten Operationen werden aber auf Objekte ausgeführt.

Objekte stellen normalerweise Klasseninstanzen dar und sind daher viel komplexere Datentypen als einfache Zahlen und Zeichen. Sie enthalten aber meist Zahlen und Zeichen als Instanz- oder Klassenvariablen.

Frage:

Keine Zeiger in Java? Wenn es in Java keine Zeiger gibt, wie kann man dann so Dinge wie verkettete Listen, wo Zeiger von einem Eintrag auf den nächsten verweisen, so daß sie diese durchqueren können, erstellen?

Antwort:

Es stimmt nicht, wenn man sagt, daß es in Java überhaupt keine Zeiger gibt - es gibt keine expliziten Zeiger. Objektreferenzen sind letztendlich Zeiger. Um eine verkettete Liste zu erzeugen, könnten Sie eine Klasse Node erstellen, die eine Instanzvariable vom Typ Node hat. Um Node-Objekte miteinander zu verketten, weisen Sie der Instanzvariablen des Objekts direkt davor in der Liste ein Node-Objekt zu. Da Objektreferenzen Zeiger sind, verhalten sich verkettete Listen, die auf diese Weise erstellt wurden, wie Sie das erwarten.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 1

Tag 5

Arrays, Bedingungen und Schleifen

Wenn Sie ein Java-Programm mit dem Wissen geschrieben haben, das Sie bis zu diesem Zeitpunkt erworben haben, wird das wahrscheinlich etwas fade sein. Wenn Sie ein Java Programm mit dem Wissen geschrieben haben, das Sie bis zu diesem Zeitpunkt erworben haben, wird das wahrscheinlich etwas fade sein.

Daß der vorige Satz zweimal hintereinander auftaucht, ist kein Fehler. Dies soll demonstrieren, wie einfach es Computer machen, die gleiche Sache immer zu wiederholen. Sie werden heute lernen, wie Sie einen Teil eines Java-Programms mit Schleifen wiederholt ausführen lassen.

Zusätzlich werden Sie lernen, wie Sie ein Programm dazu bringen, basierend auf einer vorgegebenen Logik zu entscheiden, ob es etwas tun soll. (Vielleicht würde ein Computer entscheiden, daß es nicht besonders logisch ist, in einem Buch denselben Satz zweimal in Folge zu drucken.)

Sie werden auch lernen, in Ihren Programmen Variablengruppen derselben Klasse oder desselben Datentyps in Listen, die als Arrays bezeichnet werden, zu organisieren.

Als erstes stehen die Arrays auf der heutigen Aufgabenliste. Als erstes stehen die Arrays auf der heutigen Aufgabenliste.

Arrays

Bisher hatten Sie es in den einzelnen Java-Programmen nur mit wenigen Variablen zu tun. In vielen Fällen ist es machbar, Informationen in unabhängigen Variablen zu speichern.

Was wäre allerdings, wenn Sie 20 Elemente verwandter Information hätten, die Sie alle speichern müßten? Sie könnten 20 einzelne Variablen erstellen und deren Anfangswert festlegen. Diese Vorgehensweise wird aber immer ungeeigneter, je größer die Menge der Informationen wird, mit der Sie arbeiten. Was wäre, wenn es 100 oder gar 1000 Elemente wären?

Arrays stellen eine Methode zur Speicherung einer Reihe von Elementen, die alle denselben primitiven Datentyp oder dieselbe Klasse aufweisen. Jedem Element wird innerhalb des Arrays ein eigener Speicherplatz zugewiesen. Diese Speicherplätze sind numeriert, so daß Sie auf die Informationen leicht zugreifen können.

Arrays können jede Art von Information enthalten, die auch in einer Variablen gespeichert werden kann. Sobald ein Array aber erzeugt wurde, können Sie es nur noch für diesen einen Informationstyp verwenden. Sie können z.B. ein Array für Integer, eines für String-Objekte oder eines für Arrays erzeugen. Es ist aber nicht möglich, ein Array zu erstellen, das sowohl Strings als auch Integer beinhaltet.

Java implementiert Arrays anders als andere Programmiersprachen, nämlich als Objekte, die wie andere Objekte auch behandelt werden können.

Um in Java ein Array zu erzeugen, müssen Sie folgendes tun:

1. Deklarieren einer Variablen zur Aufnahme des Arrays
2. Erstellen eines neuen Array-Objekts und Zuweisen einer Array-Variablen
3. Speichern von Elementen im Array

Deklarieren von Array-Variablen

Der erste Schritt beim Anlegen eines Arrays ist das Deklarieren einer Variablen, die das Array aufnimmt. Das geschieht genauso wie bei anderen Variablen. Array-Variablen wird ein Typ zugewiesen, den das Array aufnimmt (wie bei jeder Variablen), und der Name des Arrays. Um das Ganze von einer normalen Variablendeklaration zu unterscheiden, wird noch ein Paar leerer eckiger Klammern ([]) an den Datentyp bzw. den Klassennamen oder den Namen des Arrays angefügt.

Nachfolgend typische Deklarationen von Array-Variablen:

```
String difficultWords[];
Point hits[];
int temps[];
```

Alternativ kann eine Array-Variable definiert werden, indem die Klammern nicht nach der Variablen, sondern nach dem Typ eingefügt werden. Die obigen drei Deklarationen würden nach dieser Methode so aussehen:

```
String[] difficultWords;
Point[] hits;
int[] temps;
```

Sie werden beide Stile in Programmen sehen. Es gibt keinen Konsens darüber, welcher Stil besser lesbar ist, so daß es den persönlichen Vorlieben überlassen bleibt, eine Form zu wählen.

Erstellen von Array-Objekten

Im zweiten Schritt wird ein Array-Objekt erstellt und dieser Variablen zugewiesen. Das kann auf zwei Arten erfolgen:

- Mit dem new-Operator
- Durch direktes Initialisieren des Array-Inhalts

Da Arrays in Java Objekte sind, können Sie den Operator new verwenden, um eine neue Instanz eines Arrays zu erstellen:

```
String[] names = new String[10];
```

Diese Anweisung erstellt ein neues Array von Strings mit zehn Elementen. Beim Erstellen eines neuen Array-Objekts mit new müssen Sie angeben, wie viele Elemente das Array aufnehmen soll. Diese Anweisung fügt keine String-Objekte in das Array ein - das müssen Sie später selbst erledigen.

Array-Objekte können primitive Typen wie Ganzzahlen oder boolesche Werte genauso wie Objekte enthalten:

```
int[] temps = new int[99];
```

Beim Erstellen eines Array-Objekts mit new werden alle Elemente des Arrays automatisch initialisiert (0 für numerische Arrays, false für boolesche, '\0' für Zeichen-Arrays und null für alle anderen).

Sie können ein Array auf die gleiche Weise erstellen und initialisieren. Anstelle der Verwendung von new schließen Sie die Elemente des Arrays in Klammern ein und trennen sie durch Kommas:

```
String[] chiles = { "jalapeno", "anaheim", "serrano", "habanero", "thai" };
```



Beachten Sie bitte, daß das Schlüsselwort null von Java sich auf das Null-Objekt bezieht (und für eine beliebige Objektreferenz verwendet werden kann). Es ist allerdings nicht äquivalent mit 0 oder '0' (Null-Zeichen), wie es bei der Konstante NULL in C der Fall ist.

Alle innerhalb der Klammern stehenden Elemente müssen vom gleichen Typ sein, der dem Typ der Variablen entsprechen muß, die das Array enthält. Ein Array wird automatisch mit der Zahl der Elemente, die Sie angegeben

haben, erstellt. Dieses Beispiel erstellt ein Array mit String-Objekten namens `chiles`, das fünf Elemente enthält.

Zugreifen auf Array-Elemente

Nachdem Sie ein Array mit Anfangswerten erstellt haben, können Sie es testen und die Werte der einzelnen Zellen des Arrays ändern. Auf den Wert eines Elements in einem Array greifen Sie über den Array-Namen, gefolgt von einem Index in eckigen Klammern, zu. Diese Kombination aus Name und Index kann in Ausdrücken verwendet werden:

```
contestantScore[40] = 470;
```

Der Teil `contestantScore` dieses Ausdrucks ist eine Variable, die ein Array-Objekt beinhaltet, kann aber auch ein Ausdruck sein, der ein Array zurückgibt. Der Index legt das Element des Arrays, auf das zugegriffen wird, fest. Dies kann ebenfalls ein Ausdruck sein. Array-Indizes beginnen mit 0 wie in C und C++. Ein Array mit zehn Elementen hat folglich Index-Werte von 0 bis 9.

Alle Array-Indizes werden geprüft, um sicherzustellen, daß sie sich innerhalb der Grenzen des Arrays befinden, wie diese bei der Erzeugung des Arrays festgelegt wurden. In Java ist es unmöglich, auf einen Wert in einem Array-Element außerhalb dieser Grenzen zuzugreifen bzw. in einem solchen Element einen Wert zu speichern. Dadurch werden Probleme vermieden, wie sie beim Überschreiten von Array-Grenzen in Sprachen wie z.B. C entstehen. Betrachten Sie einmal die folgenden zwei Anweisungen:

```
String[] beatleSpeak = new String[10];
beatleSpeak[10] = "I am the eggman.";
```

Ein Programm, das die beiden vorigen Zeilen enthalten würde, würde einen Compiler-Fehler erzeugen, wenn `beatleSpeak[10]` verwendet wird. Der Fehler tritt auf, da `beatleSpeak` kein Element mit dem Index 10 hat - es verfügt zwar über 10 Elemente, die Index-Werte beginnen aber bei 0 und enden bei 9. Der Java-Compiler fängt diesen Fehler ab.

Wird der Array-Index zur Laufzeit berechnet (z.B. als Teil einer Schleife) und endet er außerhalb der Array-Grenzen, erzeugt der Java-Interpreter ebenfalls einen Fehler (um technisch korrekt zu sein, tritt eine Ausnahme auf). Sie lernen übernächste Woche am 17. Tag mehr über *Ausnahmen*.

Wie kann man verhindern, daß ein Programm versehentlich die Grenzen eines Arrays überschreitet? Sie können die Länge eines Arrays in Ihren Programmen mit der Instanzvariablen `length` testen. Sie ist für alle Array-Objekte, ungeachtet des Typs, verfügbar:

```
int len = beatleSpeak.length; // gibt 10 aus
```

Um es noch einmal zu sagen: Die Länge des Arrays ist 10, die Index-Werte gehen allerdings nur bis 9. Bei Arrays beginnt die Numerierung mit 0. Behalten Sie dies immer im Hinterkopf, wenn Sie mit Arrays arbeiten, und ziehen Sie 1 von der Länge des Arrays ab, um auf das letzte Element darin zuzugreifen.

Ändern von Array-Elementen

Wie Sie in den vorangegangenen Beispielen gesehen haben, setzen Sie einfach eine Zuweisungsanweisung hinter den Namen des Arrays mit dem Index des Elements in eckigen Klammern, um einer bestimmten Array-Zelle einen Wert zuzuweisen:

```
myGrades[4] = 85;
sentence[0] = "The";
sentence[10] = sentence[0];
```

Wichtig ist hier die Feststellung, daß ein Array mit Objekten in Java Referenzen auf diese Objekte enthält (ähnlich wie ein Pointer-Array in C oder C++). Wenn Sie einem Array-Element in einem derartigen Array einen Wert zuweisen, erstellen Sie eine Referenz auf das betreffende Objekt wie bei einer einfachen Variablen. Verschieben Sie Werte in Arrays (wie beispielsweise in der letzten Zeile oben), weisen Sie die Referenz erneut zu. Sie kopieren also nicht den Wert von einem Element in ein anderes. Arrays mit primitiven Typen wie `int` oder `float` kopieren dagegen die Werte von einem Element in ein anderes.

Arrays sind einfach zu erstellen, und ihr Inhalt ist leicht zu verändern. Für Java bieten Sie eine enorme Funktionalität. Sie werden feststellen, daß, je stärker Sie die Sprache einsetzen, Sie um so mehr mit Arrays arbeiten werden.

Um die Besprechung der Arrays abzuschließen, zeigt Listing 5.1 ein einfaches Programm, das zwei Arrays erzeugt, initialisiert, verändert und den Inhalt ausgibt.

Listing 5.1: Der gesamte Quelltext von ArrayTest.java

```

1: class ArrayTest {
2:
3:     String[] firstNames = { "Dennis", "Grace", "Bjarne", "James" };
4:     String[] lastNames = new String[firstNames.length];
5:
6:     void printNames() {
7:         int i = 0;
8:         System.out.println(firstNames[i]
9:             + " " + lastNames[i]);
10:        i++;
11:        System.out.println(firstNames[i]
12:            + " " + lastNames[i]);
13:        i++;
14:        System.out.println(firstNames[i]
15:            + " " + lastNames[i]);
16:        i++;
17:        System.out.println(firstNames[i]
18:            + " " + lastNames[i]);
19:    }
20:
21:    public static void main (String arguments[]) {
22:        ArrayTest a = new ArrayTest();
23:        a.printNames();
24:        System.out.println("-----");
25:        a.lastNames[0] = "Ritchie";
26:        a.lastNames[1] = "Hopper";
27:        a.lastNames[2] = "Stroustrup";
28:        a.lastNames[3] = "Gosling";
29:        a.printNames();
30:    }
31: }

```

Das Programm erzeugt die folgende Ausgabe:

```

Dennis null
Grace null
Bjarne null
James null
-----
Dennis Ritchie
Grace Hopper
Bjarne Stroustrup
James Gosling

```

Dieses längere Beispiel zeigt, wie Arrays erzeugt und verwendet werden. Die Klasse, die hier erstellt wird, ArrayTest, besitzt zwei Instanzvariablen, die je ein Array mit String-Objekten aufnehmen. Das erste, das den Namen firstNames trägt, wird in Zeile 3 mit vier Elementen initialisiert. Die zweite Instanzvariable, lastNames, wird in Zeile 4 deklariert. In dieses Array werden aber keine Werte eingefügt. Beachten Sie bitte, daß das Array lastNames exakt dieselbe Länge aufweist wie das Array firstNames, da hier bei der Erzeugung des Arrays die

Variable `firstNames.length` als Größenangabe für das neue Array verwendet wurde. Die Instanzvariable `length` von Array-Objekten beinhaltet die Anzahl der Elemente eines Arrays.

Die Klasse `ArrayTest` (sie besitzt auch zwei Methoden: `printNames()` und `main()`. `printNames()`), die in den Zeilen 6 bis 19 definiert wird, ist eine Hilfsmethode, die die einzelnen Elemente der Arrays `firstNames` und `lastNames` durchgeht und deren Werte anzeigt. Beachten Sie hierbei bitte, daß hier der Array-Index (`i`) anfangs auf 0 gesetzt wird, da die Numerierung bei Java-Arrays mit 0 beginnt.

Die `main()`-Methode führt schließlich die folgenden Aufgaben aus:

- In Zeile 22 wird eine Instanz der Klasse `ArrayTest` erzeugt, so daß deren Instanzvariablen und Methoden verwendet werden können.
- In Zeile 23 erfolgt dann ein Aufruf der Methode `printNames()`, den Ausgangszustand des Objekts auszugeben. Das Ergebnis sind die ersten vier Zeilen der unten abgedruckten Ausgabe. Beachten Sie, daß das Array `firstNames` initialisiert wurde, die Werte im Array `lastNames` allesamt null sind. Wenn ein Array nicht bei der Deklaration initialisiert wird, dann sind die Elemente des Arrays leer - null bei Objekten, 0 bei Zahlen und false bei booleschen Werten.
- In den Zeilen 25 bis 28 werden den Elementen im Array `lastNames` Strings zugewiesen.
- Schließlich wird in der Zeile 29 wieder die Methode `printNames()` aufgerufen, um zu zeigen, daß das Array `lastNames` inzwischen mit Werten gefüllt wurde und die Ausgabe wie erwartet funktioniert. Das Ergebnis ist in den zweiten vier Zeilen der unten abgedruckten Ausgabe wiedergegeben.



Wenn Sie die Namen in diesem Beispiel nicht erkennen, dann könnten Sie eventuell denken, daß die Autoren Referenzen auf ihre Freunde in dieses Buch einbringen. Alle Namen sind Namen von Erfindern, die die führenden Entwickler von Computer-Programmiersprachen sind: Dennis Ritchie (C), Bjarne Stroustrup (C++), Grace Hopper (COBOL) und James Gosling (Java).

Eine letzte Anmerkung gilt es zu Listing 5.1 zu machen: Es ist ein Beispiel für furchtbar schlechten Programmierstil. Normalerweise verwenden Sie Schleifen, um die einzelnen Elemente eines Arrays durchzugehen, anstatt jedes Element einzeln zu behandeln. Dadurch wird der Code wesentlich kürzer und in vielen Fällen einfacher zu lesen. Wenn Sie später am heutigen Tag die Schleifen kennenlernen, werden Sie eine überarbeitete Version des aktuellen Beispiels sehen.

Mehrdimensionale Arrays

Wenn Sie Arrays bereits in anderen Programmiersprachen verwendet haben, werden Sie sich vielleicht fragen, ob Java mehrdimensionale Arrays unterstützt. Dies sind Arrays, die über mehr als einen Index verfügen, um mehrere Dimensionen zu repräsentieren.

Mehrere Dimensionen sind z.B. praktisch, um eine (x,y)-Tabelle als Array anzulegen.

In Java werden multidimensionale Arrays nicht unterstützt. Allerdings können Sie ein Array mit Arrays (die wiederum Arrays enthalten können usw., über beliebig viele Dimensionen) deklarieren:

```
int coords[] [] = new int[12][12];
coords[0][0] = 1;
coords[0][1] = 2;
```

Blockanweisungen

Anweisungen werden in Java in Blocks zusammengefaßt. Der Anfang und das Ende eines Blocks werden mit geschweiften Klammern (`{}`) gekennzeichnet.

Sie haben bis hierher bereits Blocks in den Programmen für die folgenden Aufgaben verwendet:

- Für Methoden und Variablen in einer Klassendefinition
- Um die Anweisungen zu kennzeichnen, die zu einer Methode gehören

Blocks werden auch als Blockanweisungen bezeichnet, da ein gesamter Block überall dort verwendet werden könnte, wo auch eine einzelne Anweisung verwendet werden kann. Die Anweisungen in einem Block werden von oben nach unten ausgeführt.

Blocks können sich innerhalb anderer Blocks befinden, wie das der Fall ist, wenn Sie eine Methode in eine Klassendefinition einfügen.

Einen wichtigen Punkt gibt es zu Blocks anzumerken: Ein Block erzeugt einen sogenannten Gültigkeitsbereich für lokale Variablen, die in dem Block erzeugt werden.



Gültigkeitsbereich ist in der Programmierung der Begriff für den Teil eines Programms, in dem eine Variable existiert und verwendet werden kann. Wenn das Programm den Gültigkeitsbereich einer Variablen verläßt, dann existiert diese nicht, und es treten Fehler auf bei dem Versuch, auf diese zuzugreifen.

Der Gültigkeitsbereich einer Variablen ist der Block, in dem sie erzeugt wurde. Wenn Sie in einem Block lokale Variablen deklarieren und verwenden, dann hören diese Variablen auf zu existieren, sobald der Block ausgeführt ist. Die folgende Methode `testBlock()` beinhaltet z.B. einen Block:

```
void testBlock() {
    int x = 10;
    { // Beginn des Blocks
        int y = 40;
        y = y + x;
    } // Ende des Blocks
}
```

In dieser Methode werden zwei Variablen definiert: `x` und `y`. Der Gültigkeitsbereich der Variablen `y` ist der Block, in dem sie sich befindet, und sie kann auch nur in diesem angesprochen werden. Wenn Sie versuchen würden, sie in einem anderen Teil der Methode `testBlock()` zu verwenden, würde ein Fehler auftreten. Die Variable `x` wurde innerhalb der Methode, aber außerhalb des inneren Blocks definiert, so daß sie in der gesamten Methode angesprochen werden kann.

Blöcke werden normalerweise aber nicht auf diese Weise - allein in einer Methodendefinition - verwendet, wie das im vorigen Beispiel der Fall war. Vorwiegend haben Sie bisher Blöcke gesehen, die Klassen- und Methodendefinitionen umgeben. Daneben gibt es eine andere häufige Verwendung für Blockanweisungen in logischen Strukturen und Schleifen, über die Sie später in der heutigen Lektion mehr erfahren.

if-Bedingungen

Einer der wesentlichen Aspekte bei der Programmierung ist die Möglichkeit für ein Programm, zu entscheiden, was es tun soll. Dies wird durch eine bestimmte Anweisungsart, die Bedingung genannt wird, erreicht.



Eine *Bedingung* ist eine Programmanweisung, die nur dann ausgeführt wird, wenn eine bestimmte Situation eintritt.

Die elementarste Bedingung wird mit dem Schlüsselwort `if` erzeugt. Eine `if`-Bedingung verwendet einen

booleschen Ausdruck, um zu entscheiden, ob eine Anweisung ausgeführt werden soll. Wenn der Ausdruck `true` zurückliefert, wird die Anweisung ausgeführt.

Hier nun ein einfaches Beispiel, das die Nachricht "You call that a haircut?" (zu deutsch: »Sie bezeichnen das als einen Haarschnitt?«) nur unter einer Bedingung ausgibt: Der Wert der Variablen `age` ist größer als 39:

```
if (age > 39)
    System.out.println("You call that a haircut?");
```

Wenn Sie wollen, daß etwas passiert, wenn der boolesche Ausdruck `false` zurückgibt, dann verwenden Sie das optionale Schlüsselwort `else`. Das folgende Beispiel verwendet sowohl `if` als auch `else`:

```
if (blindDateIsAttractive == true)
    restaurant = "Benihana's";
else
    restaurant = "Burrito Hut";
```

Die `if`-Anweisung führt hier anhand des Ergebnisses des booleschen Ausdrucks in der `if`-Anweisung unterschiedliche Anweisungen aus.



Zwischen den `if`-Anweisungen von Java und C/C++ gibt es folgenden Unterschied: Der Testausdruck in der `if`-Anweisung muß in Java einen booleschen Wert zurückgeben (`true` oder `false`). In C kann der Test auch einen Integer zurückgeben.

Wenn Sie eine `if`-Anweisung verwenden, ist es möglich, lediglich eine einzige Anweisung nach dem Testausdruck ausführen zu lassen (im vorigen Beispiel wurde der Variablen `restaurant` ein Wert zugewiesen). Da aber ein Block überall dort verwendet werden kann, wo auch eine einzelne Anweisung stehen kann, können Sie, wenn Sie mehr als nur eine Anweisung ausführen wollen (was normalerweise der Fall ist), diese Anweisungen zwischen ein Paar geschweiffter Klammern setzen. Sehen Sie sich einmal das folgende Code-Schnipsel an, das eine Erweiterung des `Jabberwock`-Objekts von Tag 2 ist:

```
if (attitude == "angry" ) {
    System.out.println("The jabberwock is angry.");
    System.out.println ("Have you made out a will?");
} else {
    System.out.println ("The jabberwock is in a good mood.");
    if (hungry)
        System.out.println("It still is hungry, though.");
    else System.out.println("It wanders off.");
}
```

In diesem Beispiel wird der Testausdruck (`attitude == "angry"`) verwendet. Dadurch soll festgestellt werden, ob das `Jabberwock` zornig oder in guter Stimmung ist. Wenn das `Jabberwock` guter Dinge ist, wird mit dem Testausdruck (`hungry`) geprüft, ob das `Jabberwock` hungrig ist - davon ausgehend, daß man ein hungriges `Jabberwock`, auch wenn es gut gelaunt ist, meiden sollte. Die Bedingung `if (hungry)` ist eine andere Form des Ausdrucks `if (hungry == true)`. Für boolesche Testausdrücke dieser Art, ist es bei der Programmierung üblich, den letzten Teil des Ausdrucks wegzulassen.

Listing 5.2 ist ein weiteres einfaches Beispiel - diesmal in Form einer vollständigen Applikation. Die Klasse `EvenSteven` beinhaltet eine Hilfsmethode mit dem Namen `checkEven()`, die überprüft, ob ein Wert gerade ist. Ist dies der Fall, gibt sie `Steven` auf den Bildschirm aus.

Listing 5.2: Der gesamte Quelltext von `EvenSteven.java`

```
1: class EvenSteven {
2:
3:     void evenCheck(int val) {
```



```

4:         System.out.println("Value is "
5:             + val + ". ");
6:         if (val % 2 == 0)
7:             System.out.println("Steven!");
8:     }
9:
10:    public static void main (String arguments[]) {
11:        EvenSteven e = new EvenSteven();
12:
13:        e.evenCheck(1);
14:        e.evenCheck(2);
15:        e.evenCheck(54);
16:        e.evenCheck(77);
17:        e.evenCheck(1346);
18:    }
19: }

```

Das Programm liefert die folgende Ausgabe:

```

Value is 1.
Value is 2.
Steven!
Value is 54.
Steven!
Value is 77.
Value is 1346.
Steven!

```

Das Herz der Klasse EvenSteven ist die Methode checkEven() (Zeilen 3 bis 8). Hier werden die Werte geprüft und die entsprechenden Nachrichten ausgegeben. Anders als bei den Methoden, die Sie in den vorangegangenen Beispielen definiert haben, verwendet die Methode checkEven() ein Integer-Argument (siehe Zeile 3). Die Methode checkEven() beginnt damit, den Wert, der ihr übergeben wurde, auszugeben. Anschließend wird das Argument mit einer if-Bedingung getestet, um zu sehen, ob es sich um eine gerade Zahl handelt.

Der Test mit dem Modulo-Operator ergibt, wie Sie sich sicherlich von Tag 3 her erinnern werden, den Rest der Division der Operanden. Wenn der Rest der Division einer Zahl durch 2 0 ergibt, dann ist diese Zahl gerade.

Wenn eine Zahl gerade ist, wird "Steven!" angezeigt (Sie lernen morgen mehr darüber, wie Sie Methoden mit Argumenten definieren).

Die main()-Methode dieser Applikation erzeugt eine neue Instanz der Klasse EvenSteven und prüft diese, indem sie die Methode checkEven() wiederholt mit unterschiedlichen Werten aufruft. In der Ausgabe haben nur die geraden Werte die Nachricht "Steven!".

Der Bedingungsoperator

Eine Alternative zur Verwendung der Schlüsselwörter if und else in einer Bedingungsanweisung ist der *Bedingungsoperator*, der auch *ternärer Operator* genannt wird. Der *Bedingungsoperator* ist ein *ternärer Operator*, weil er drei Teile umfaßt.

Der Bedingungsoperator ist ein Ausdruck, was bedeutet, daß er einen Wert zurückgibt (im Gegensatz zum allgemeineren if, das zur Ausführung einer beliebigen Anweisung oder eines Blocks führen kann). Der Bedingungsoperator ist besonders für sehr kurze oder einfache Bedingungen nützlich und sieht so aus:

```
Test ? trueresult : falseresult
```

Das Wort Test ist ein Ausdruck, der true oder false ausgibt, wie beim Testen in der if-Anweisung. Ist der Test true, gibt der Bedingungsoperator den Wert von trueresult aus. Ist er false, gibt er den Wert von falseresult aus. Folgende Bedingung prüft z.B. die Werte von myScore und yourScore, gibt den kleineren der beiden Werte aus

und weist diesen Wert der Variablen `ourBestScore` zu:

```
int ourBestScore = myScore > yourScore ? myScore : yourScore;
```

Diese Zeile mit dem Bedingungsoperator entspricht dem folgenden If-Else-Konstrukt:

```
if (myScore > yourScore)
    ourBestScore = myScore;
else
    ourBestScore = yourScore;
```

Der Bedingungsoperator hat eine sehr niedrige Präzedenz. Das bedeutet, daß er normalerweise erst nach allen Unterausdrücken ausgewertet wird. Nur die Zuweisungsoperatoren haben eine noch niedrigere Präzedenz. Zum Auffrischen des Gedächtnisses können Sie die Präzedenz von Operatoren in der Tabelle 3.6 von Tag 3 nachschlagen.

switch-Bedingungen

Eine übliche Vorgehensweise beim Programmieren in jeder Sprache ist das Testen einer Variablen auf einen bestimmten Wert. Falls sie nicht zu diesem Wert paßt, wird sie anhand eines anderen Wertes geprüft, und falls dieser wieder nicht paßt, wird wieder mit einem anderen Wert geprüft usw. Werden nur if-Anweisungen verwendet, kann das je nachdem, wie der Quelltext dabei formatiert wurde und wie viele verschiedene Optionen geprüft werden müssen, sehr unhandlich sein. Zum Schluß erhalten Sie if- Anweisungen wie im folgenden Beispiel oder noch unhandlicher:

```
if (oper == '+')
    addargs(arg1,arg2);
else if (oper == '=')
    subargs(arg1,arg2);
else if (oper == '*')
    multargs(arg1,arg2);
else if (oper == '/')
    divargs(arg1,arg2);
```

Diese Form nennt man *verschachtelte if-Anweisungen*, weil jede else-Anweisung ihrerseits weitere if-Anweisungen enthält, bis alle möglichen Tests ausgeführt wurden.

Eine übliche Kurzform anstelle verschachtelter if-Anweisungen ermöglicht Ihnen in manchen Fällen, Tests und Aktionen gemeinsam in einer Anweisung auszuführen. Das ist die Anweisung `switch` oder `case`. In Java ist es die Anweisung `switch`, die sich wie in C verhält:

```
switch (grade) {
    case 'A':
        System.out.println("Great job -- an A!");
        break;
    case 'B':
        System.out.println("Good job -- a B!");
        break;
    case 'C':
        System.out.println("Your grade was a C.");
        break;
    default: System.out.println("An F -- consider cheating!");
}
```

Die `switch`-Anweisung basiert auf einem Test; im vorigen Beispiel wird der Wert der Variablen gerade getestet. Die Testvariable, die einen der primitiven Typen `byte`, `char`, `short` oder `int` aufweisen kann, wird nacheinander mit jedem der `case`-Werte verglichen. Wenn eine Übereinstimmung gefunden wird, wird (werden) die Anweisung(en) im Anschluß an den `case`-Wert ausgeführt. Wenn keine Übereinstimmung gefunden wird, wird (werden) die Anweisung(en) des `default`-Zweiges ausgeführt. Sollte dieser nicht vorhanden sein und wird auch keine

Übereinstimmung gefunden, dann wird die switch-Anweisung beendet, ohne daß irgend etwas ausgeführt wird.

Die Java-Implementierung von switch ist einigen Einschränkungen unterworfen - der Testwert und die Werte der case-Zweige dürfen nur primitive Typen aufweisen, die sich in den Typ int casten lassen. Sie können keine größeren primitiven Typen verwenden, wie z.B. long oder float, Strings oder andere Objekte. Auch können Sie nur auf Gleichheit und keine anderen Relationen testen. Diese Grenzen beschränken die switch-Anweisung auf sehr einfache Fälle. Im Gegensatz dazu können verschachtelte if-Anweisungen mit allen Testmöglichkeiten und allen Datentypen arbeiten.

Hier noch einmal das obige Beispiel, diesmal jedoch mit einer switch-Anweisung anstelle verschachtelter if-Anweisungen:

```
switch (oper) {
case '+':
    addargs(arg1, arg2);
    break;
case '*':
    subargs(arg1, arg2);
    break;
case '-':
    multargs(arg1, arg2);
    break;
case '/':
    divargs(arg1, arg2);
    break;
}
```

In diesem Beispiel gibt es zwei Dinge, die Sie beachten sollten: Erstens können Sie nach jeder case-Anweisung eine einzelne Anweisung oder so viele Anweisungen, wie eben nötig, schreiben. Anders als bei der if-Anweisung müssen Sie keine geschweiften Klammern verwenden, wenn Sie mehrere Anweisungen angeben. Zweitens sollten Sie die break-Anweisung, die sich in diesem Beispiel in jedem der case-Zweige befindet, beachten. Ohne die break-Anweisung würden, sobald eine Übereinstimmung gefunden wird, alle Anweisungen für diese Übereinstimmung und alle Anweisungen, die diesen Anweisungen folgen, ausgeführt. Dies geht so lange, bis eine break-Anweisung auftaucht oder das Ende der switch-Anweisung erreicht ist. In einigen Fällen kann dies genau das sein, was Sie wollen. In den meisten Fällen sollten Sie eine break-Anweisung einfügen, um sicherzustellen, daß nur der richtige Code ausgeführt wird. Die break-Anweisung, worüber Sie im Abschnitt »Schleifen verlassen« mehr lernen werden, bricht die Ausführung an der aktuellen Stelle ab und springt zu dem Code im Anschluß an die nächste schließende geschweifte Klammer (}).

Eine gute Anwendung, bei der Sie die break-Anweisung weglassen, ist, wenn mehrere Werte die gleiche(n) Anweisung(en) ausführen sollen. In diesem Fall können Sie mehrere case-Zweige ohne Anweisungen eingeben, dann führt switch die ersten Anweisungen aus, die es findet. In der folgenden switch-Anweisung wird beispielsweise die Zeichenkette "x is an even number." ausgegeben, wenn x einen geraden Wert, z.B. 2, 4, 6 oder 8, hat. Alle anderen Werte von x geben die Zeichenkette "x is an odd number." aus.

```
switch (x) {
case 2:
case 4:
case 6:
case 8:
    System.out.println("x is an even number.");
    break;
default: System.out.println("x is an odd number.");
}
```

Das Listing 5.3 zeigt noch ein weiteres Beispiel für die Verwendung der switch-Anweisung. Diese Klasse mit dem Namen NumberReader konvertiert Ziffern in das englische Wort, das der Ziffer entspricht. Dazu wird die Methode convertIt() verwendet.

Listing 5.3: Der gesamte Quelltext von NumberReader.java

```

1: class NumberReader {
2:
3:     String convertNum(int val) {
4:         switch (val) {
5:             case 0: return "zero ";
6:             case 1: return "one ";
7:             case 2: return "two ";
8:             case 3: return "three ";
9:             case 4: return "four ";
10:            case 5: return "five ";
11:            case 6: return "six ";
12:            case 7: return "seven ";
13:            case 8: return "eight ";
14:            case 9: return "nine ";
15:            default: return " ";
16:        }
17:    }
18:
19:    public static void main (String arguments[]) {
20:        NumberReader n = new NumberReader();
21:        String num = n.convertNum(4) + n.convertNum(1) + n.convertNum(3);
22:        System.out.println("413 converts to " + num);
23:    }
24: }

```

Das Programm liefert die folgende Ausgabe:

```
413 converts to four one three
```

Das Herz dieses Beispiels ist natürlich die switch-Anweisung in der Mitte der Methode convertNum() in den Zeilen 4 bis 16. Diese switch-Anweisung nimmt das Integer-Argument, das convertNum() übergeben wurde, und gibt, wenn sie eine Übereinstimmung findet, den entsprechenden String zurück. Dies steht im Gegensatz zu den anderen Methoden, die Sie bis zu diesem Zeitpunkt verwendet und die keinen Wert zurückgegeben haben. (Über diesen Punkt werden Sie morgen mehr lernen.)

In dem Programm NumberReader werden keine break-Anweisungen benötigt, statt dessen wird die return-Anweisung verwendet. return ist break ähnlich mit der Ausnahme, daß return die Methode komplett verläßt und einen Wert zurückgibt. Auch darüber erfahren Sie morgen mehr, wenn Sie lernen, wie Sie Methoden definieren.

Inzwischen haben Sie wahrscheinlich schon genügend main()-Methoden gesehen, um zu verstehen, was hier passiert. Trotzdem wollen wir diese hier kurz durchgehen.

In Zeile 20 wird eine neue Instanz der Klasse NumberReader erzeugt.

In Zeile 21 wird eine String-Variable mit dem Namen num definiert. Diese wird den verketteten String-Wert dreier Ziffern aufnehmen. Jede der Ziffern wird über einen Aufruf der Methode convertNum() konvertiert.

Und schließlich zeigt der Code in der Zeile 22 das Ergebnis an.

for-Schleifen

Wie in C wiederholt die for-Schleife eine Anweisung oder einen Anweisungsblock mehrmals, bis eine Bedingung zutrifft. for-Schleifen werden häufig für einfache Wiederholungen verwendet, um Blockanweisungen mehrmals auszuführen und dann zu stoppen. Sie können for-Schleifen für jede Schleifenart verwenden.

Die for-Schleife sieht in Java so aus:

```
for (Initialisierung; Test; Inkrement) {
Anweisungen;
}
```

Der Beginn der for-Schleife hat drei Teile:

- Initialisierung ist ein Ausdruck, der den Beginn der Schleife einleitet. Wenn Sie einen Schleifenindex verwenden, kann er durch diesen Ausdruck deklariert und initialisiert werden, z.B. `int i = 0`. Die Variablen, die Sie in diesem Teil der for-Schleife deklarieren, sind lokal in bezug auf die Schleife. Das bedeutet, daß sie zur Schleife gehören und nach der vollständigen Ausführung der Schleife nicht mehr existieren (anders als bei C oder C++). Sie können in diesem Bereich mehr als eine Variable initialisieren, indem Sie die einzelnen Ausdrücke durch Kommas voneinander trennen. Die Anweisung `int i = 0, int j = 10` würde hier die Variablen `i` und `j` deklarieren. Beide Variablen wären in bezug auf die Schleife lokal.
- Test ist der Ausdruck, der nach jeder Iteration der Schleife ausgeführt wird. Der Test muß ein boolescher Ausdruck oder eine Funktion sein, die einen booleschen Wert zurückgibt, z.B. `i < 10`. Ergibt der Test `true`, wird die Schleife ausgeführt. Sobald er `false` ergibt, wird die Schleifenausführung gestoppt.
- Inkrement ist ein beliebiger Ausdruck oder Funktionsaufruf. Üblicherweise wird Inkrement verwendet, um den Wert des Schleifenindex zu ändern, um ihn näher an den Endwert heranzubringen und damit zur Ausgabe von `false` und zur Beendigung der Schleife zu sorgen. Wie schon im Initialisierungsbereich können Sie mehr als einen Ausdruck unterbringen, wenn Sie die einzelnen Ausdrücke mit Kommas voneinander trennen.

Der Anweisungen-Teil der for-Schleife enthält die Anweisungen, die bei jeder Wiederholung der Schleife ausgeführt werden. Wie bei `if` können Sie hier entweder eine einzelne Anweisung oder einen Block einbinden. Im vorherigen Beispiel wurde ein Block benutzt, weil das üblich ist. Im nächsten Beispiel einer for-Schleife wird allen Elementen eines String-Arrays der Wert "Mr." zugewiesen:

```
String[] salutation = new String[10];
int i; // Schleifenindex

for (i = 0; i < salutation.length; i++)
    salutation[i] = "Mr.";
```

In diesem Beispiel überwacht die Variable `i`, wie oft die Schleife bereits durchlaufen wurde. Gleichzeitig stellt sie auf bequeme Weise einen Index für das Array dar. Wir starten die for-Schleife mit `i = 0`. Der Test, ob die Schleife beendet werden soll, prüft, ob der aktuelle Index kleiner als die Länge des Arrays ist (sobald der Index größer als das Array lang ist, sollten Sie die Schleife beenden). Das Inkrement fügt dem Index bei jedem Schleifendurchlauf 1 hinzu. So können Sie bei jeder Iteration der Schleife dem Element mit dem aktuellen Index den Wert "Mr." zuweisen.

Jeder Teil der for-Schleifendeklaration kann eine leere Anweisung sein, d.h., Sie können einfach ein Semikolon ohne Ausdruck oder Anweisung eingeben, dann wird dieser Teil der for-Schleife ignoriert. Wenn Sie in einer for-Schleife eine leere Anweisung verwenden, müssen Sie eventuell Schleifenvariablen oder Schleifenindizes manuell anderswo im Programm initialisieren oder inkrementieren.

Auch im Körper der for-Schleife kann eine leere Anweisung stehen, falls alles, was Sie bezwecken, in der ersten Zeile der Schleife steht. Im folgenden Beispiel wird die erste Primzahl gesucht, die größer ist als 4000 (dafür wird in der Schleifendeklaration die Methode `notPrime()` aufgerufen, die in der Lage sein soll zu prüfen, ob eine Zahl eine Primzahl ist):

```
for (i = 4001; notPrime(i); i += 2)
;
```

Beachten Sie, daß ein häufig in C gemachter Fehler auch in Java passieren kann: Nach der ersten Zeile der for-Schleife wird versehentlich ein Semikolon eingegeben:

```
for (i = 0; i < 10; i++);
    x = x * i; // diese Zeile befindet sich nicht innerhalb der Schleife
```


Das erste Semikolon beendet die Schleife mit einer leeren Anweisung, ohne daß $x = x * i$ in der Schleife ausgeführt wird. Die Zeile $x = x * i$ wird nur einmal ausgeführt, weil sie sich außerhalb der for-Schleife befindet. Achten Sie darauf, daß Ihnen dieser Fehler in Ihren Java-Programmen nicht passiert.

Am Ende des Abschnitts über die for-Schleifen wollen wir das Beispiel mit den Namen aus dem Abschnitt über die Arrays neu schreiben. Der Original-Quelltext ist lang und wiederholt sich. Außerdem arbeitet das Programm in dieser Version nur mit einem Array mit vier Elementen. Diese Version (siehe Listing 5.4) ist kürzer und wesentlich flexibler (erzeugt aber dieselbe Ausgabe).

Listing 5.4: Der gesamte Quelltext von NamesLoop.java

```

1: class NamesLoop {
2:
3:     String[] firstNames = { "Dennis", "Grace", "Bjarne", "James" };
4:     String[] lastNames = new String[firstNames.length];
5:
6:     void printNames() {
7:         for (int i = 0; i < firstNames.length; i++)
8:             System.out.println(firstNames[i] + " " + lastNames[i]);
9:     }
10:
11:     public static void main (String arguments[]) {
12:         NamesLoop a = new NamesLoop();
13:         a.printNames();
14:         System.out.println("-----");
15:         a.lastNames[0] = "Ritchie";
16:         a.lastNames[1] = "Hopper";
17:         a.lastNames[2] = "Stroustrup";
18:         a.lastNames[3] = "Gosling";
19:
20:         a.printNames();
21:     }
22: }

```

Das Program erzeugt die folgende Ausgabe:

```

Dennis null
Grace null
Bjarne null
James null
-----
Dennis Ritchie
Grace Hopper
Bjarne Stroustrup
James Gosling

```

Der einzige Unterschied zwischen diesem Beispiel und dem Listing 5.1 findet sich in der Methode printNames(). Anstatt die Elemente des Arrays einzeln anzusprechen, verwendet dieses Beispiel eine for-Schleife, um das Array Element für Element durchzugehen und beim letzten Element zu stoppen. Indem Sie eine etwas allgemeiner gestaltete Schleife verwenden, haben Sie die Möglichkeit, die printNames()-Methode für jedes Array mit beliebiger Größe zu verwenden und gleichzeitig alle darin befindlichen Elemente auszugeben.

while- und do-Schleifen

Nun bleiben noch die Schleifenarten while und do zu erlernen. Wie mit for-Schleifen kann mit while- und do-Schleifen ein Java-Code-Block wiederholt ausgeführt werden, bis eine bestimmte Bedingung erfüllt ist. Welche der drei Schleifenarten (for, while oder do) Sie bevorzugt verwenden, ist eine Sache des persönlichen

Programmierstils.

Die Schleifen `while` und `do` sind mit denen in C und C++ identisch, mit dem Unterschied, daß die Testbedingung in Java boolesch sein muß.

while-Schleifen

Die `while`-Schleife wird zum Wiederholen einer Anweisung oder von Blockanweisungen verwendet, solange eine bestimmte Bedingung zutrifft. Im Anschluß sehen Sie ein Beispiel für eine `while`-Schleife:

```
while (i < 10) {
    x = x * i++; // Rumpf der Schleife
}
```

Die Bedingung, die das Schlüsselwort `while` begleitet, ist ein boolescher Ausdruck - im vorigen Beispiel `i < 10`. Wenn der Ausdruck `true` ergibt, dann führt die `while`-Schleife den Rumpf der Schleife aus und prüft anschließend die Bedingung erneut. Dieser Prozeß wiederholt sich so lange, bis die Bedingung `false` ergibt. Obwohl die obige Schleife ein Paar geschweifeter Klammern für eine Blockanweisung im Rumpf der Schleife verwendet, wären diese hier nicht nötig, da sich nur eine Anweisung darin befindet - `x = x * i++`. Die Klammern stellen aber kein Problem dar. Benötigt werden sie, wenn Sie später eine weitere Anweisung in den Rumpf der Schleife einfügen.

Das Listing 5.5 zeigt ein Beispiel für eine `while`-Schleife, die die Elemente eines Arrays mit Ganzzahlen (in `array1`) in ein Array mit Gleitpunktzahlen (in `array2`) kopiert. Dabei werden alle Elemente nacheinander in `float` konvertiert. Ist eines der Elemente im ersten Array 1, wird die Schleife sofort an diesem Punkt beendet.

Listing 5.5: Der gesamte Quelltext von `CopyArrayWhile.java`

```
1: class CopyArrayWhile {
2:     public static void main (String arguments[]) {
3:         int[] array1 = { 7, 4, 8, 1, 4, 1, 4 };
4:         float[] array2 = new float[array1.length];
5:
6:         System.out.print("array1: [ ");
7:         for (int i = 0; i < array1.length; i++) {
8:             System.out.print(array1[i] + " ");
9:         }
10:        System.out.println("]");
11:
12:        System.out.print("array2: [ ");
13:        int count = 0;
14:        while ( count < array1.length && array1[count] != 1) {
15:            array2[count] = (float) array1[count];
16:            System.out.print(array2[count++] + " ");
17:        }
18:        System.out.println("]");
19:    }
20: }
```

Das Programm erzeugt die folgende Ausgabe:

```
array1: [ 7 4 8 1 4 1 4 ]
array2: [ 7.0 4.0 8.0 ]
```

Lassen Sie uns nun einen Blick darauf werfen, was in der `main()`-Methode vor sich geht:

Die Zeilen 3 und 4 deklarieren die Arrays. `array1` ist ein Array für `int`-Werte, das ich mit irgendwelchen geeigneten Werten initialisiert habe. `array2` ist vom Typ `float`, hat dieselbe Größe wie `array1`, wird aber nicht mit Werten initialisiert.

Die Zeilen 6 bis 10 dienen zur Ausgabe. Hier wird einfach `array1` mit einer `for`-Schleife durchgegangen, und die

einzelnen Werte darin werden auf dem Bildschirm ausgegeben.

In den Zeilen 13 bis 17 passieren die interessanten Dinge. Die Anweisung hier weist zugleich array2 die Werte zu (dafür werden die int-Werte in float-Werte konvertiert) und gibt sie auf dem Bildschirm aus. Wir beginnen mit der Variablen count, die den Array-Index darstellt. Der Testausdruck in der while-Schleife prüft zwei Bedingungen: Zum einen wird überwacht, ob das Ende des Arrays erreicht ist, und zum anderen, ob einer der Werte in array1 1 ist (wie Sie sich erinnern werden, war dies Teil der Beschreibung des Programms). Der Testausdruck läßt sich mit dem logischen UND-Operator && formen. Denken Sie bitte daran, daß der Operator && sicherstellt, daß beide Bedingungen true sind, bevor der gesamte Ausdruck true ergibt. Wenn einer davon false ist, ergibt der gesamte Ausdruck false, und die Schleife endet. Was passiert nun in diesem speziellen Beispiel? Die Ausgabe zeigt, daß die ersten vier Elemente von array1 in array2 kopiert wurden. Allerdings befand sich mitten unter den Werten eine 1, die die Ausführung der Schleife beendete. Ohne das Auftreten einer 1 sollte array2 am Ende dieselben Elemente beinhalten wie array1.

Falls die Bedingung beim ersten Durchlauf false ist (z.B. wenn das erste Element im ersten Array 1 ist), wird der Körper der while-Schleife nie ausgeführt. Soll die Schleife mindestens einmal ausgeführt werden, haben Sie folgende Möglichkeiten:

- Sie duplizieren den Schleifenkörper und fügen ihn außerhalb der while-Schleife ein.
- Sie verwenden eine do-Schleife (wird unten beschrieben).

Die do-Schleife gilt als bessere Lösung der zwei Möglichkeiten.

do...while-Schleifen

Die do-Schleife entspricht der while-Schleife, außer daß sie eine bestimmte Anweisung oder einen Block so oft ausführt, bis eine Bedingung false ergibt. Auf den ersten Blick erscheinen die beiden Schleifentypen gleich. Der wesentliche Unterschied ist, daß while-Schleifen die Bedingung vor der Ausführung der Schleife prüfen, so daß der Schleifenkörper unter Umständen nie ausgeführt wird, wenn die Bedingung beim ersten Durchlauf false ist, während bei do-Schleifen der Schleifenkörper mindestens einmal vor dem Testen der Bedingung ausgeführt wird.

Der Unterschied ist in etwa so, wie beim Ausleihen des Autos der Eltern. Wenn man die Eltern fragt, bevor man das Auto ausleiht, und sie sagen nein, dann hat man das Auto nicht. Fragt man sie dagegen, nachdem man sich das Auto ausgeliehen hat, und sie sagen nein, dann hatte man das Auto bereits.

do-Schleifen sehen in Java so aus:

```
do {
    x = x * i++; // Rumpf der Schleife
} while (i < 10);
```

Der Rumpf der Schleife wird einmal ausgeführt, bevor die Bedingung $i < 10$ ausgewertet wird. Wenn der Test anschließend true ergibt, wird die Schleife erneut ausgeführt. Wenn der Test allerdings false ergibt, dann wird die Schleife beendet. Denken Sie immer daran, daß bei do-Schleifen der Rumpf der Schleife mindestens einmal ausgeführt wird.

Listing 5.6 zeigt ein einfaches Beispiel einer do-Schleife, die eine Nachricht bei jedem Durchlauf der Schleife ausgibt (in diesem Beispiel zehnmal):

Listing 5.6: Der gesamte Quelltext von DoTest.java

```
1: class DoTest {
2:     public static void main (String arguments[]) {
3:         int x = 1;
4:
5:         do {
6:             System.out.println("Looping, round " + x);
7:             x++;
8:         } while (x <= 10);
9:     }
```

```
10: }
```

Im folgenden die Ausgabe des Programms:

```
Looping, round 1
Looping, round 2
Looping, round 3
Looping, round 4
Looping, round 5
Looping, round 6
Looping, round 7
Looping, round 8
Looping, round 9
Looping, round 10
```

Unterbrechen von Schleifen

Alle Schleifen (for, while und do) enden, wenn die geprüfte Bedingung erfüllt ist. Was passiert, wenn etwas Bestimmtes im Schleifenkörper stattfindet und Sie die Schleife bzw. die aktuelle Iteration frühzeitig beenden wollen? Hierfür können Sie die Schlüsselwörter break und continue verwenden.

Sie haben break bereits als Teil der switch-Anweisung kennengelernt. break stoppt die Ausführung von switch, und das Programm läuft weiter. Bei Verwendung mit einer Schleife bewirkt das Schlüsselwort break das gleiche - es hält die Ausführung der aktiven Schleife sofort an. Enthalten Schleifen verschachtelte Schleifen, wird die Ausführung mit der nächstäußeren Schleife wieder aufgenommen. Andernfalls wird die Ausführung des Programms ab der nächsten Anweisung nach der Schleife fortgesetzt.

Nehmen wir als Beispiel an, Sie haben eine while-Schleife, die Elemente von einem Array in ein anderes kopiert. Jedes Element im Array soll kopiert werden, bis das Ende des Arrays erreicht ist oder bis ein Element 1 ist. Sie können den zweiten Fall im while-Körper testen und dann break verwenden, um die Schleife zu beenden:

```
while (count < array1.length) {
if (array1[count] == 1) {
break;
}
array2[count] = (float) array1[count++];
}
```

continue verhält sich ähnlich wie break, außer, daß die Ausführung der Schleife nicht komplett gestoppt wird, sondern mit der nächsten Iteration erneut beginnt. Bei do- und while-Schleifen bedeutet das, daß die Ausführung des Blocks erneut beginnt. Bei for-Schleifen wird der Ausdruck Inkrement ausgewertet, dann wird der Block ausgeführt. continue ist nützlich, wenn Sie spezielle Fälle in einer Schleife berücksichtigen wollen. In dem vorherigen Beispiel, in dem ein Array in ein anderes kopiert wird, können Sie testen, ob das aktuelle Element 1 ist, und die Schleife neu starten, falls dies zutrifft. Dadurch erreichen Sie, daß das resultierende Array nie eine 1 enthält. Da dabei Elemente im ersten Array übersprungen werden, müssen Sie folglich die zwei verschiedenen Array-Zähler überwachen:

```
int count = 0;
int count2 = 0;
while (count < array1.length) {
if (array1[count] == 1){
count++;
continue;
}
array2[count2++] = (float)array1[count++];
}
```

Benannte Schleifen

Sowohl `break` als auch `continue` kann optional beschriftet werden, um Java mitzuteilen, wo genau die Ausführung wieder aufgenommen werden soll. Ohne Beschriftung springt `break` aus der innersten Schleife (zu der umgebenden Schleife oder zur nächsten Anweisung außerhalb der Schleife), während `continue` mit der nächsten Iteration des in Klammern stehenden Schleifenrumpfs beginnt. Durch Verwendung beschrifteter `break`- und `continue`-Anweisungen können Sie die Ausführung außerhalb von verschachtelten Schleifen oder eine Schleife außerhalb der aktiven Schleife fortsetzen.

Um eine Schleife zu benennen, fügen Sie vor dem Anfangsteil der Schleife eine Beschriftung (*Label*) und einen Doppelpunkt ein. Wenn Sie dann `break` oder `continue` verwenden, fügen Sie den Namen der Beschriftung direkt nach dem Schlüsselwort ein:

```
out:
    for (int i = 0; i < 10; i++) {
        while (x < 50) {
            if (i * x++ > 400)
                break out;
            // innere Schleife
        }
        // äußere Schleife
    }
```

In diesem Codeteil wird die äußere `for`-Schleife mit `out` bezeichnet. Ein `break` innerhalb der `for`- und der `while`-Schleife veranlaßt die Unterbrechung beider Schleifen, falls eine bestimmte Bedingung in beiden Schleifen erfüllt ist, und startet wieder ab der Beschriftung (`out`).

Das folgende kleine Programm in Listing 5.7 ist ein weiteres Beispiel mit einer verschachtelten `for`-Schleife. Beide Schleifen werden sofort beendet, wenn die Summe der zwei Zähler in der innersten Schleife größer ist als vier:

Listing 5.7: Der gesamte Quelltext von `LabelTest.java`

```
1: class LabelTest {
2:     public static void main (String arguments[]) {
3:
4:         thisLoop:
5:             for (int i = 1; i <= 5; i++)
6:                 for (int j = 1; j <= 3; j++) {
7:                     System.out.println("i is " + i + ", j is " + j);
8:                     if (( i + j) > 4)
9:                         break thisLoop;
10:                }
11:             System.out.println("end of loops");
12:         }
13: }
```

Die Ausgabe dieses Programms ist wie folgt:

```
i is 1, j is 1
i is 1, j is 2
i is 1, j is 3
i is 2, j is 1
i is 2, j is 2
i is 2, j is 3
end of loops
```

Wie Sie sehen, wird die Schleife so oft wiederholt, bis die Summe von `i` und `j` größer ist als 4, dann enden beide Schleifen, führen zum äußeren Block zurück und die endgültige Meldung wird ausgegeben.

Zusammenfassung

Sie haben gelernt, wie eine Array-Variable deklariert wird, wie ein Array-Objekt erstellt und dieser Variablen zugewiesen wird und wie Sie auf Elemente in einem Array zugreifen und diese ändern.

Zu Bedingungen zählen die Anweisungen `if` und `switch`, mit denen Sie auf der Grundlage eines booleschen Tests in andere Teile eines Programms verzweigen können.

Ferner haben Sie die Schleifen `for`, `while` und `do` gelernt. Mit allen drei Schleifenarten können Sie einen Programmteil wiederholt ausführen, bis eine bestimmte Bedingung erfüllt ist.

Das muß wiederholt werden:

- Sie werden diese drei Features häufig in Ihren Programmen verwenden.
- Sie werden diese drei Features häufig in Ihren Programmen verwenden.

Fragen und Antworten

Frage:

Ich habe in einer Blockanweisung für eine `if`-Anweisung eine Variable deklariert. Als die `if`-Anweisung verarbeitet war, verschwand die Definition dieser Variablen. Wie kann das sein?

Antwort:

Vom technischen Gesichtspunkt bilden Blockanweisungen innerhalb von Klammern einen neuen Gültigkeitsbereich. Das bedeutet, daß eine in einem Block deklarierte Variable nur innerhalb dieses Blocks sichtbar und nutzbar ist. Sobald die Ausführung des Blocks abgeschlossen ist, verschwinden alle Variablen, die Sie darin deklariert haben.

Es empfiehlt sich, Variablen im äußersten Block, in dem sie gebraucht werden, zu deklarieren. Das ist normalerweise am Anfang einer Blockanweisung. Eine Ausnahme hierzu können sehr einfache Variablen sein, z.B. Indexzähler in `for`-Schleifen, deren Deklaration in der ersten Zeile der `for`-Schleife eine einfache Kurzform ist.

Frage:

Warum kann man `switch` nicht mit Strings benutzen?

Antwort:

Strings sind Objekte, während `switch` in Java nur auf die primitiven Typen angewandt werden kann, die in Ganzzahlen konvertiert werden können (`byte`, `char`, `short` und `int`). Zum Vergleichen von Strings müssen Sie verschachtelte `if`-Anweisungen verwenden. Damit sind auch Vergleiche von Strings möglich.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).

Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 1

Tag 6

Java: Eine Klassesprache

Wenn Sie von einer anderen Programmiersprache zu Java wechseln, dann werden Sie eventuell mit den Klassen etwas zu kämpfen haben. Der Begriff *Klasse* scheint zum einen synonym mit dem Begriff *Programm* zu sein, zum anderen könnten Sie aber unsicher sein, was die Beziehung zwischen den Begriffen anbelangt.

In Java besteht ein Programm aus einer Hauptklasse und beliebig vielen Hilfsklassen, die zur Unterstützung der Hauptklasse benötigt werden. Diese Hilfsklassen beinhalten beliebige Klassen aus der Java-Klassenbibliothek, die Sie eventuell brauchen (wie z.B. String, Math etc.).

Heute werden Sie in bezug darauf, was Sie über dieses Thema wissen, den Gipfel der Klassen erklimmen. Im einzelnen werden Sie über folgendes lesen:

- Die Teile einer Klassendefinition
- Deklarieren und Verwenden von Instanzvariablen
- Definieren und Verwenden von Methoden
- Die main()-Methode, die in Java-Applikationen verwendet wird

Definieren von Klassen

Da Sie in jedem der vorangegangenen Kapitel bereits Klassen erstellt haben, sollten Sie mit den Grundlagen der Definition von Klassen bereits vertraut sein. Um eine Klasse zu definieren, verwenden Sie das Schlüsselwort `class` und den Namen der Klasse:

```
class Ticker {  
    // Rumpf der Klasse  
}
```

Standardmäßig werden Klassen von der Klasse `Object` abgeleitet. Diese ist die Superklasse aller Klassen in der Klassenhierarchie von Java.

Wenn Ihre Klasse eine Subklasse einer anderen Klasse ist, dann wird über das Schlüsselwort `extends` die Superklasse der neuen Klasse angegeben. Sehen Sie sich hierzu die folgende Subklasse der Klasse `Ticker` an:

```
class SportsTicker extends Ticker {  
    // Rumpf der Klasse  
}
```

Erstellen von Instanz- und Klassenvariablen

Wenn Sie eine Klasse von einer Superklasse ableiten, dann werden Sie Verhaltensweisen und Eigenschaften hinzufügen wollen, die die neue Klasse von der Klasse, von der sie abgeleitet wurde, unterscheidet.

Diese Verhaltensweisen und Eigenschaften definieren Sie über die Variablen und Methoden der neuen Klasse. In diesem Abschnitt arbeiten Sie mit drei verschiedenen Variablentypen: Klassenvariablen, Instanzvariablen und lokalen Variablen. Der darauffolgende Abschnitt behandelt Methoden.

Definieren von Instanzvariablen

Am 3. Tag haben Sie gelernt, wie lokale Variablen, d.h. Variablen in Methodendefinitionen, deklariert und initialisiert werden. Instanzvariablen werden zum Glück auf genau die gleiche Weise deklariert und definiert. Der wesentliche Unterschied ist ihre Position in der Klassendefinition. Instanzvariablen gelten als solche, wenn sie außerhalb einer Methodendefinition deklariert werden. Üblicherweise werden die meisten Instanzvariablen direkt nach der ersten Zeile der Klassendefinition definiert. Listing 6.1 zeigt als Beispiel eine einfache Definition für eine Klasse namens `Jabberwock`, die

von der Klasse `Reptile` abgeleitet wird. Diese Klassendefinition enthält vier Instanzvariablen:

Listing 6.1: Die komplette Quelltext von `Jabberwock.java`

```
1: class Jabberwock extends Reptile {
2:
3:     String color;
4:     String sex;
5:     boolean hungry;
6:     int age;
7: }
```

Diese Klassendefinition umfaßt vier Variablen. Da diese Variablen nicht innerhalb einer Methode definiert sind, handelt es sich um Instanzvariablen. Die folgenden Variablen befinden sich in der Klasse

- `color`: die Farbe des Jabberwock, z.B. Orange oder Zitronengelb
- `sex`: ein String, der das Geschlecht des Jabberwock angibt
- `hungry`: eine boolesche Variable, die `true` ist, wenn das Jabberwock hungrig ist, ansonsten ist sie `false`
- `age`: das Alter des Jabberwock in Jahren

Konstanten

Variablen sind sehr nützlich, wenn Sie Informationen speichern wollen, die man zur Laufzeit eines Programms ändern können soll. Soll sich der Wert allerdings zur Laufzeit eines Programms nicht ändern, können Sie einen speziellen Variablentyp verwenden: die Konstanten.



Eine *Konstante* ist eine Variable, deren Wert sich nie ändert (was im Zusammenhang mit dem Wort »Variable« seltsam erscheinen mag).

Konstanten sind sehr nützlich für die Definition von Werten, die allen Methoden eines Objekts zur Verfügung stehen sollen. Mit anderen Worten kann man mit Konstanten unveränderlichen, objektweit genutzten Werten einen aussagekräftigen Namen geben. In Java können Sie mit allen Variablenarten Konstanten erzeugen: Instanzvariablen, Klassenvariablen und lokalen Variablen.



Konstante lokale Variablen waren in Java 1.02 nicht möglich, wurden aber mit Java 1.1 in der Sprache eingeführt. Dies wird wichtig, wenn Sie ein Applet erstellen, das zu Java 1.02 völlig kompatibel sein soll. Über dieses Thema lernen Sie in Woche 2 mehr.

Um eine Konstante zu deklarieren, benutzen Sie das Schlüsselwort `final` vor der Variablendeklaration und geben für diese Variable einen Anfangswert ein:

```
final float pi = 3.141592;
final boolean debug = false;
final int numberOfJenny = 8675309;
```

Konstanten sind auch nützlich zur Benennung verschiedener Zustände eines Objekts, das dann auf diese Zustände getestet werden kann. Nehmen wir beispielsweise an, Sie brauchen einen Schriftzug, der links, rechts oder zentriert ausgerichtet werden kann. Diese Werte können Sie als konstante Ganzzahlen definieren:

```
final int LEFT = 0;
final int RIGHT = 1;
final int CENTER = 2;
```

Die Variable `alignment` (Ausrichtung) wird dann ebenfalls als `int` deklariert, um darin die aktuelle Ausrichtung zu speichern:

```
int alignment;
```

Danach können Sie im Körper einer Methodendefinition eine der Ausrichtungen setzen:

```
this.alignment = CENTER;
```

oder auf eine bestimmte Ausrichtung prüfen:

```
switch (this.alignment) {
case LEFT: // Ausrichtung Links verarbeiten
...
break
case RIGHT: // Ausrichtung Rechts verarbeiten
...
break
case CENTER: // Ausrichtung Zentriert verarbeiten
...
break
}
```

Konstanten machen es oft leichter, ein Programm zu verstehen. Um diesen Punkt zu verdeutlichen, sollten Sie sich einmal die beiden folgenden Anweisungen ansehen und vergleichen, welche mehr über ihre Funktion aussagt:

```
this.alignment = CENTER;
this.alignment = 2;
```

Klassenvariablen

Wie Sie in den vorherigen Lektionen gelernt haben, sind Klassenvariablen global innerhalb einer bestimmten Klasse und für alle Instanzen der jeweiligen Klasse gültig.

Klassenvariablen eignen sich gut zur Kommunikation zwischen verschiedenen Objekten der gleichen Klasse oder zum Verfolgen globaler Zustände in bestimmten Objekten.

Um eine Klassenvariable zu deklarieren, benutzen Sie das Schlüsselwort `static` in der Klassendeklaration:

```
static int sum;
static final int maxObjects = 10;
```

Erstellen von Methoden

Am 4. Tag haben Sie gelernt, daß Methoden das Verhalten eines Objekts bestimmen, d.h. was passiert, wenn das Objekt erstellt wird, und welche Operationen es während seiner Lebenszeit ausführen kann.

In dieser Lektion erhalten Sie eine Grundeinführung in die Methodendefinition und wie Methoden funktionieren. Morgen steigen Sie in mehr Einzelheiten über die Dinge ein, die Sie mit Methoden anstellen können.

Definieren von Methoden

Die Definition von Methoden besteht aus folgenden vier Teilen:

- Name der Methode
- Objekttyp oder der primitive Typ, den die Methode zurückgibt
- Liste der Parameter
- Methodenrumpf

Die ersten drei Teile bilden die sogenannte Signatur der Methode.



Um die heutige Lektion nicht unnötig zu verkomplizieren, habe ich zwei optionale Teile der Definition einer Methode

weggelassen: Modifier, z.B. public oder private, und das Schlüsselwort throws, das die Ausnahmen bezeichnet, die eine Methode auswerfen kann. Sie lernen diese Teile der Methodendefinition in Woche 3.

In anderen Sprachen genügt der Name der Methode (bzw. der Funktion, Subroutine oder Prozedur), um sie von anderen im Programm vorhandenen Methoden zu unterscheiden.

In Java sind mehrere Methoden mit dem gleichen Namen möglich, jedoch mit einem anderen Rückgabetyt und einer anderen Argumentenliste. Dieses sogenannte Überladen von Methoden lernen Sie morgen kennen.

Eine einfache Methodendefinition sieht in Java so aus:

```
Rückgabetyt Methodenname (Typ1 Arg1, Typ2 Arg2, Typ3 Arg3...) {
//Rumpf der Methode
}
```

Rückgabetyt ist der primitive Typ oder die Klasse des Wertes, den die Methode zurückgibt. Das kann einer der primitiven Typen, ein Klassenname oder void sein, falls die Methode keinen Wert zurückgibt.

Gibt diese Methode ein Array-Objekt zurück, können die Array-Klammern entweder nach dem Rückgabetyt oder nach der Parameterliste eingegeben werden. Da die erste Art wesentlich übersichtlicher ist, wird sie in den heutigen Beispielen (und im gesamten Buch) angewandt:

```
int[] makeRange (int lower, int upper) {...}
```

Bei der Parameterliste einer Methode handelt es sich um verschiedene Variablendeklarationen, die durch Kommas getrennt werden und zwischen Klammern stehen. Diese Parameter bilden lokale Variablen im Körper der Methode, deren Werte Objekte oder Werte von Primitivtypen sind, die beim Aufrufen der Methode übergeben werden.

Im Methodenkörper können Anweisungen, Ausdrücke, Methodenaufrufe anderer Objekte, Bedingungen, Schleifen usw. stehen - alles, was Sie in den bisherigen Lektionen gelernt haben.

Hat Ihre Methode einen Rückgabetyt (d.h. der Rückgabetyt wurde nicht mit void deklariert), muß irgendwo im Methodenkörper ein Wert zurückgegeben werden. Sie verwenden hierfür das Schlüsselwort return.

Listing 6.2 zeigt ein Beispiel einer Klasse, die die Methode makeRange() definiert. makeRange() nimmt zwei Ganzzahlen entgegen - eine obere und eine untere Grenze - und erstellt ein Array, das alle zwischen diesen Grenzen liegenden Ganzzahlen (einschließlich der Grenzwerte) enthält.

Listing 6.2: Der gesamte Quelltext von RangeClass.java

```
1: class RangeClass {
2:     int[] makeRange(int lower, int upper) {
3:         int arr[] = new int[ (upper - lower) + 1 ];
4:
5:         for (int i = 0; i < arr.length; i++) {
6:             arr[i] = lower++;
7:         }
8:         return arr;
9:     }
10:
11:     public static void main(String arguments[]) {
12:         int theArray[];
13:         RangeClass theRange = new RangeClass();
14:
15:         theArray = theRange.makeRange(1, 10);
16:         System.out.print("The array: [ ");
17:         for (int i = 0; i < theArray.length; i++) {
18:             System.out.print(theArray[i] + " ");
19:         }
20:         System.out.println("]");
21:     }
22:
23: }
```

Die Ausgabe dieses Programms sieht so aus:

The array: [1 2 3 4 5 6 7 8 9 10]

Die Methode main() in dieser Klasse testet die Methode makeRange() durch Anlegen eines Bereichs, wobei die obere und untere Grenze des Bereichs 1 bzw. 10 ist (siehe Zeile 6). Dann wird eine for-Schleife benutzt, um die Werte des neuen Arrays auszugeben.

Das this-Schlüsselwort

Sicherlich möchten Sie im Körper einer Methodendefinition einmal auf das aktuelle Objekt verweisen, beispielsweise, um auf Instanzvariablen des Objekts zu verweisen oder das aktuelle Objekt als Argument an eine andere Methode weiterzugeben.

Um auf das aktuelle Objekt in diesen Fällen Bezug zu nehmen, können Sie das Schlüsselwort this verwenden.

Sie können es an jeder beliebigen Stelle eingeben, an der das Objekt erscheinen kann, z.B. in einer Punkt-Notation, um auf Instanzvariablen des Objekts zu verweisen, oder als Argument für eine Methode, als Rückgabewert der aktuellen Methode usw. Hier ein paar Beispiele für die Verwendung des Schlüsselwortes this:

```
t = this.x; // Instanzvariable x für dieses Objekt
this.resetData(this); // Aufruf der in dieser Klasse definierten
// resetData-Methode, an die das aktuelle Objekt
// übergeben wird
return this; // Rückgabe des aktuellen Objekts
```

In vielen Fällen können Sie eventuell das Schlüsselwort this weglassen. Sie können sich auf Instanzvariablen und Methodenaufrufe, die in der aktuellen Klasse definiert sind, einfach auch über den Namen beziehen. this ist in diesen Referenzen implizit vorhanden. Die ersten zwei Beispiele könnten somit auch wie folgt geschrieben werden:

```
t = x; // Instanzvariable x für dieses Objekt
resetData(this); // Aufruf der in dieser Klasse definierten
// resetData-Methode
```



Ob Sie das Schlüsselwort this für Instanzvariablen weglassen können, hängt davon ab, ob es Variablen mit dem gleichen Namen in dem aktuellen Gültigkeitsbereich gibt. Einzelheiten hierzu finden Sie im nächsten Abschnitt.

Da this eine Referenz auf die aktuelle Instanz einer Klasse ist, ist es sinnvoll, das Schlüsselwort nur innerhalb der Definition einer Instanzmethode zu verwenden. Klassenmethoden, d.h. Methoden, die mit dem Schlüsselwort static deklariert sind, können this nicht verwenden.

Gültigkeitsbereich von Variablen und Methodendefinitionen

Was Sie über eine Variable unbedingt wissen müssen, um sie verwenden zu können, ist deren Gültigkeitsbereich.



Der *Gültigkeitsbereich* einer Variablen legt fest, wo eine Variable verwendet werden kann.

Wenn Sie eine Variable deklarieren, hat diese immer einen eingeschränkten Gültigkeitsbereich. Der Gültigkeitsbereich einer Variablen legt fest, wo diese Variable verwendet werden kann. Variablen mit einem lokalem Gültigkeitsbereich können z.B. nur in dem Block verwendet werden, in dem sie definiert wurden. Der Gültigkeitsbereich von Instanzvariablen bezieht die gesamte Klasse ein, so daß diese von Methoden innerhalb der Klasse verwendet werden können.

Wenn Sie sich auf eine Variable in einer Methodendefinition beziehen, sucht Java zuerst eine Definition dieser Variablen im aktuellen Gültigkeitsbereich (der ein Block sein kann), dann durchsucht es den äußeren Gültigkeitsbereich bis zur Definition der aktuellen Methode. Ist die gesuchte Variable keine lokale Variable, sucht Java nach einer Definition dieser Variablen als Instanzvariable in der aktuellen Klasse und zum Schluß in jeder Superklasse.

Aufgrund der Art, in der Java nach dem Bereich einer bestimmten Variablen sucht, können Sie eine Variable in einem niedrigeren Bereich erstellen, so daß eine Definition der gleichen Variablen den Originalwert der Variablen »verbirgt«. Das kann aber zu Fehlern führen, die schwer zu finden sind.

Betrachten Sie z.B. dieses kleine Java-Programm:

Listing 6.3: Der gesamte Quelltext von ScopeTest.java

```

1: class ScopeTest {
2:     int test = 10;
3:
4:     void printTest () {
5:         int test = 20;
6:         System.out.println("test = " + test);
7:     }
8:
9:     public static void main(String arguments[]) {
10:         ScopeTest st = new ScopeTest();
11:         st.printTest();
12:     }
13: }
```

Das Programm erzeugt die folgende Ausgabe:

```
test = 20
```

Diese Klasse hat zwei Variablen mit dem gleichen Namen und der gleichen Definition: Die erste, eine Instanzvariable, hat den Namen test und ist auf den Wert 10 initialisiert. Die zweite ist eine lokale Variable mit dem gleichen Namen, jedoch dem Wert 20. Die lokale Variable in test in der Methode printTest() verbirgt die Instanzvariable test. Die Methode printTest() innerhalb der von main() gibt aus diesem Grund test = 20 aus.

Sie können dies Problem umgehen, indem Sie this.test verwenden, um sich spezifisch auf die Instanzvariable zu beziehen, oder nur test, um sich auf die lokale Variable zu beziehen.

Der Konflikt wird also vermieden, indem Sie sich auf die Variable über den Gültigkeitsbereich des Objekts beziehen.

Eine heimtückischere Situation tritt ein, wenn Sie eine Variable in einer Subklasse, die bereits in einer Superklasse vorkommt, neu definieren. Das kann sehr komplizierte Fehler im Code verursachen. Beispielsweise werden Methoden aufgerufen, die den Wert einer Instanzvariablen ändern sollen, nun jedoch die falsche ändern. Ein anderer Fehler kann auftreten, wenn ein Objekt von einer Klasse in eine andere konvertiert wird. Eventuell wird dabei der Wert einer Instanzvariablen auf geheimnisvolle Weise geändert (da sie den Wert von der Superklasse, nicht von der beabsichtigten Klasse entnommen hat).

Die beste Art, dieses Verhalten zu vermeiden, ist, sicherzustellen, daß Sie beim Definieren von Variablen in einer Subklasse die Variablen in allen Superklassen dieser Klasse kennen und nichts duplizieren, was sich bereits dort befindet.

Argumente an Methoden übergeben

Wenn Sie eine Methode mit Objektparametern aufrufen, werden die Variablen, die Sie an den Körper der Methode übergeben, als Referenz übergeben. Das bedeutet, daß sich alles, was Sie mit diesen Objekten in der Methode anstellen, gleichermaßen auf die Originalobjekte auswirkt. Dies beinhaltet Arrays und alle Objekte, die in Arrays enthalten sind. Wenn Sie ein Array an eine Methode übergeben und dann seinen Inhalt ändern, wirkt sich das auch auf das Original-Array aus. (Beachten Sie, daß die Primitivtypen als Wert weitergegeben werden.)

Listing 6.4: Die PassByReference-Klasse

```

1: class PassByReference {
2:     int onetoZero(int arg[]) {
3:         int count = 0;
4:
5:         for (int i = 0; i < arg.length; i++) {
6:             if (arg[i] == 1) {
7:                 count++;
8:                 arg[i] = 0;
```

```

9:         }
10:    }
11:    return count;
12: }
13: public static void main(String arguments[]) {
14:     int arr[] = { 1, 3, 4, 5, 1, 1, 7 };
15:     PassByReference test = new PassByReference();
16:     int numOnes;
17:
18:     System.out.print("Values of the array: [ ");
19:     for (int i = 0; i < arr.length; i++) {
20:         System.out.print(arr[i] + " ");
21:     }
22:     System.out.println("]");
23:
24:     numOnes = test.onetoZero(arr);
25:     System.out.println("Number of Ones = " + numOnes);
26:     System.out.print("New values of the array: [ ");
27:     for (int i = 0; i < arr.length; i++) {
28:         System.out.print(arr[i] + " ");
29:     }
30:     System.out.println("]");
31: }
32: }

```

Das Programm erzeugt die folgende Ausgabe:

```

Values of the array: [ 1 3 4 5 1 1 7 ]
Number of Ones = 3
New values of the array: [ 0 3 4 5 0 0 7 ]

```

Beachten Sie bitte die Definition der Methode `onetoZero()` in den Zeilen 2 bis 12, die als Argument lediglich ein Array erwartet. Die Methode `onetoZero()` bewirkt zwei Dinge:

- Sie zählt die Anzahl von Einsen im Array und gibt diesen Wert aus.
- Sie sucht Einsen und ersetzt sie im Array durch Nullen.

Die `main()`-Methode in der Klasse `PassByReference` testet die Methode `onetoZero()`. Wir wollen nun die Methode `main()` Zeile für Zeile durchgehen, um zu sehen, was da passiert.

- In Zeilen 14 bis 16 werden die anfänglichen Variablen für dieses Beispiel eingerichtet. Bei der ersten handelt es sich um ein Ganzzahlen-Array. Die zweite Variable ist eine Instanz der Klasse `PassByReference`, die im Variablentest gespeichert ist. Die dritte ist eine einfache Ganzzahl, die die Anzahl von im Array vorkommenden Einsen aufnehmen soll.
- Die Zeilen 18 bis 22 geben die Anfangswerte des Arrays aus. Sie können die Ausgabe dieser Zeilen in der ersten Zeile des Ausgabeabschnitts erkennen.
- In Zeile 24 findet die eigentliche Arbeit statt: Hier wird die Methode `onetoZero()`, die im Objekt `test` definiert ist, aufgerufen und an das in `arr` gespeicherte Array weitergereicht. Diese Methode gibt die Zahl von Einsen im Array aus, die dann der Variablen `numOnes` zugewiesen wird.
- Zeile 25 gibt die Zahl von Einsen aus, d.h. den Wert, den die Methode `onetoZero()` ermittelt hat. Hier ist das Ergebnis, wie zu erwarten war, 3.
- Die übrigen Zeilen geben die Array-Werte aus. Da eine Referenz des Array-Objekts an die Methode abgegeben wird, ändert sich auch das Original-Array, wenn das Array innerhalb dieser Methode geändert wird. Die Ausgabe der Werte in den Zeilen 27 bis 30 beweist das. In der letzten Ausgabezeile ist ersichtlich, daß alle Einsen im Array in Nullen geändert wurden.

Klassenmethoden

Die Beziehung zwischen Klassen- und Instanzvariablen läßt sich direkt mit der zwischen Klassen- und Instanzmethoden vergleichen.

Klassenmethoden sind für alle Instanzen der Klasse selbst verfügbar und können anderen Klassen zur Verfügung gestellt werden. Außerdem ist bei Klassenmethoden im Gegensatz zu Instanzmethoden keine Instanz der Klasse nötig, damit deren Klassenmethoden aufgerufen werden können.

Die Java-Klassenbibliothek beinhaltet z.B. eine Klasse mit dem Namen `Math`. Die Klasse `Math` definiert eine Reihe von mathematischen Operationen, die Sie in jedem beliebigen Programm bzw. auf jeden der Zahlentypen verwenden können, wie das im folgenden der Fall ist:

```
float root = Math.sqrt(453.0);
System.out.print("The larger of x und y is" + Math.max(x,y));
```

Um Klassenmethoden zu definieren, benutzen Sie das Schlüsselwort `static` vor der Methodendefinition, wie beim Erstellen von Klassenvariablen. Die Klassenmethode `max` könnte beispielsweise folgende Signatur haben:

```
static int max (int arg1, int arg2) { ... }
```

Auf ähnliche Weise liefert Java *Wrapper-Klassen* oder auch *Hüllklassen* für alle Grundtypen, z.B. die Klassen `Integer`, `Float` und `Boolean`. Anhand der in diesen Klassen definierten Klassenmethoden können Sie Objekte in Grundtypen und umgekehrt konvertieren.

Die Klassenmethode `parseInt()` z.B. in der Klasse `Integer` arbeitet mit Strings zusammen. Ein String wird als Argument an die Methode geschickt. Dieser String wird zur Ermittlung eines Rückgabewertes verwendet, der als `int` zurückgegeben wird:

```
int count = Integer.parseInt("42") // Ergibt 42
```

In der obigen Anweisung wird der String `"42"` von der Methode `parseInt()` als `Integer` mit dem Wert 42 zurückgegeben. Dieser Wert wird in der Variablen `count` gespeichert.

Beindet sich das Schlüsselwort `static` nicht vor dem Namen einer Methode, so wird diese zur Instanzmethode. Instanzmethoden beziehen sich immer auf ein konkretes Objekt anstatt auf die Klasse selbst. Am 2. Tag haben Sie eine Instanzmethode erstellt, die `feedJabberwock()` hieß und ein einzelnes `Jabberwock`-Objekt gefüttert hat.

Die meisten Methoden, die auf ein bestimmtes Objekt anwendbar sind oder sich auf ein Objekt auswirken, sollten als Instanzmethoden definiert werden. Methoden, die eine gewisse allgemeine Nützlichkeit bieten und sich nicht direkt auf eine Instanz einer Klasse auswirken, werden bevorzugt als Klassenmethoden deklariert.

Entwickeln von Java-Applikationen

Sie haben gelernt, Klassen, Objekte, Klassen- und Instanzvariablen sowie Methoden zu erstellen. Alles, was Sie noch lernen müssen, um etwas Lauffähiges zu produzieren, ist das Schreiben einer Java-Applikation.

Anwendungen, um Ihr Gedächtnis aufzufrischen, sind Java-Programme, die eigenständig laufen. Applikationen unterscheiden sich von Applets, für die HotJava oder ein Java-fähiger Browser benötigt wird, um sie ausführen zu können. Ein Großteil dessen, was Sie in den bisherigen Lektionen durchgearbeitet haben, waren Java-Applikationen. Nächste Woche tauchen Sie in die Entwicklung von Applets ein. (Um Applets schreiben zu können, ist mehr Basiswissen erforderlich, denn Applets müssen auch mit dem Browser interagieren und das Grafiksystem zum Zeichnen und Aktualisieren des Ausgegebenen verwenden. Das alles lernen Sie nächste Woche.)

Eine Java-Applikation besteht aus einer oder mehreren Klassen und kann beliebig umfangreich sein. HotJava ist ein Beispiel einer Java-Applikation. Obwohl die Java-Applikationen, die Sie bis jetzt erzeugt haben, nichts anderes tun, als Zeichen auf dem Bildschirm oder in ein Fenster auszugeben, können Sie auch Applikationen erstellen, die Fenster, Grafik und Elemente der Benutzerschnittstelle verwenden, wie das auch bei Applets der Fall ist.

Das einzige, was Sie brauchen, um eine Java-Applikation auszuführen, ist eine Klasse, die als »Sprungbrett« für den Rest Ihres Java-Programms dient. Ist Ihr Programm eher klein, kann eine Klasse unter Umständen genügen.

Die Klasse, die das Sprungbrett für Ihr Java-Programm bildet, muß nur eines haben: eine `main()`-Methode. Wenn Sie Ihre kompilierte Java-Klasse (mit dem Java-Interpreter) ausführen, ist die Methode `main()` das erste, was aufgerufen wird. Das dürfte für Sie keine Überraschung mehr sein, da Sie ja in den bisherigen Lektionen mehrmals Java-Applikationen mit einer `main()`-Methode erstellt haben.

Die Signatur der Methode `main()` sieht immer so aus:

```
public static void main (String arg[]) {
```



```
//Rumpf der Methode
```

```
}
```

Die einzelnen Teile von `main()` haben folgende Bedeutung:

- `public` bedeutet, daß diese Methode für andere Klassen und Objekte verfügbar ist. Die `main()`-Methode muß als `public` deklariert werden. Sie lernen mehr über `public` und `private` in Woche 3.
- `static` bedeutet, daß es sich um eine Klassenmethode handelt.
- `void` bedeutet, daß die `main()`-Methode keinen Wert zurückgibt.
- `main()` erhält einen Parameter: ein `String`-Array. Dieses Argument dient für Befehlszeilenargumente (das lernen Sie im nächsten Abschnitt).

In den Rumpf der `main()`-Methode kann jeder beliebige Code eingefügt werden, der benötigt wird, um eine Anwendung zu starten: Variablen oder Instanzen von Klassen, die Sie deklariert haben.

Bei der Ausführung der `main()`-Methode bleibt zu berücksichtigen, daß es sich um eine Klassenmethode handelt. Deshalb wird die Klasse, die sie beinhaltet, nicht automatisch beim Ablauf des Programms ausgeführt. Soll diese Klasse als Objekt behandelt werden, müssen Sie direkt in der `main()`-Methode eine Instanz der Klasse erstellen (bei allen bisherigen Beispielen ist das der Fall).

Hilfsklassen

Ihre Java-Applikation kann entweder nur aus einer Klasse, wie das bei den meisten größeren Programmen der Fall ist, oder aus mehreren Klassen bestehen. Dabei werden dann verschiedene Instanzen der einzelnen Klassen erzeugt und verwendet, während das Programm ausgeführt wird. Sie können so viele Klassen erzeugen, wie Sie wollen, und solange sich diese in dem Verzeichnis befinden, auf das die Umgebungsvariable `CLASSPATH` zeigt, ist Java auch in der Lage, diese zu finden, während Ihr Programm läuft, sofern Sie das JDK verwenden.

Beachten Sie bitte, daß nur in einer Klasse, der Start-Klasse, eine `main()`-Methode vorhanden sein muß. Denken Sie bitte daran, daß `main()` nur verwendet wird, um das Programm zu starten und ein Startobjekt zu erzeugen. Danach sind die Methoden in den verschiedenen Klassen und Objekten an der Reihe. Obwohl Sie `main()`-Methoden in Hilfsklassen implementieren können, werden diese ignoriert, wenn das Programm ausgeführt wird.

Java-Anwendungen und Befehlszeilenargumente

Da Java-Anwendungen in sich geschlossene Programme sind, sollte man in der Lage sein, Argumente oder Optionen an dieses Programm weiterzugeben, um festzulegen, wie das Programm abläuft, oder um ein allgemeines Programm mit verschiedenen Eingabearten zu betreiben. Befehlszeilenargumente können für viele verschiedene Zwecke benutzt werden, z.B. um die Debugging-Eingabe zu aktivieren, den Namen einer Datei zu bezeichnen, die gelesen oder beschrieben werden soll, oder für andere Informationen, die Ihr Java-Programm wissen soll.

Argumente an Java-Programme übergeben

Wie Sie Argumente an eine Java-Applikation übergeben, hängt von der Plattform ab, auf der Sie Java ausführen. Unter Windows und Unix können Sie Argumente über die Kommandozeile übergeben. Auf dem Macintosh stellt Ihnen der Java Runner ein spezielles Fenster zur Verfügung, um diese Argumente einzugeben.

Windows/Solaris

Um Argumente an ein Programm unter Windows und Solaris zu übergeben, geben Sie diese auf Kommandozeilenebene mit dem Aufruf Ihres Java-Programms an:

```
java Myprogram argumentEins 2 drei
```

In dem vorigen Beispiel wurden drei Argumente an das Programm übergeben: `argument1`, die Zahl 2 und drei. Beachten Sie bitte, daß ein Leerzeichen die einzelnen Argumente voneinander trennt.

Um Argumente zu gruppieren, die ihrerseits Leerzeichen enthalten, schließen Sie diese in doppelte Anführungszeichen ein. Das Argument "No Shoes No Shirt No Service" stellt für ein Programm ein einziges Argument dar, da durch die Anführungszeichen verhindert wird, daß die Leerzeichen zur Trennung der einzelnen Argumente verwendet werden. Die Anführungszeichen sind nicht in dem Argument enthalten, wenn dieses an das Programm geschickt und von der `main()`-Methode verarbeitet wird.

Argumente in einer Java-Applikation verarbeiten

Wie verarbeitet Java Argumente? Java speichert sie in einem String-Array, das an die Methode `main()` in Ihrer Java-Applikation weitergegeben wird. Sie erinnern sich an die Signatur von `main()`:

```
public static void main (String arg[]) { ... }
```

Hier ist `arg` der Name des String-Arrays, das die Argumentenliste enthält. Sie können dieses Array nach Geschmack benennen.

Innerhalb Ihrer `main()`-Methode können Sie dann die Argumente, die Ihr Programm erhalten hat, beliebig handhaben, indem Sie das Array entsprechend durchgehen. Das Beispiel in Listing 6.5 ist eine sehr einfache Klasse, die die erhaltenen Argumente zeilenweise ausgibt.

Listing 6.5: Der gesamte Quelltext von `EchoArgs.java`

```
1: class EchoArgs {
2:     public static void main(String arguments[]) {
3:         for (int i = 0; i < arguments.length; i++) {
4:             System.out.println("Argument " + i + ": " + arguments[i]);
5:         }
6:     }
7: }
```

Im Anschluß sehen Sie ein Beispiel für Argumente, die dieses Programm verarbeitet:

```
java EchoArgs Wilhelm Niekro Hough 49
```

Wenn Sie die `EchoArgs`-Applikation mit den obigen Kommandozeilenargumenten ausführen, wird die folgende Ausgabe erzeugt:

```
Argument 0: Wilhelm
```

```
Argument 1: Niekro
```

```
Argument 2: Hough
```

```
Argument 3: 49
```

Hier nun ein weiteres Beispiel

```
java EchoArgs "Hoyt Wilhelm" Charlie Hough
```

```
Argument 0: Hoyt Wilhelm
```

```
Argument 1: Charlie
```

```
Argument 2: Hough
```

Beachten Sie die Gruppierung der Argumente im zweiten Beispiel. Die Anführungszeichen um `Hoyt Wilhelm` sorgen dafür, daß das Argument als eine Einheit innerhalb des Argumenten-Arrays behandelt wird.

Ein wichtiger Faktor der Argumente, die Sie an ein Java-Programm weitergeben, ist, daß alle Argumente in einem String-Array gespeichert werden. Sollen sie nicht als Strings behandelt werden, müssen Sie sie in den gewünschten Typ umwandeln.



Das Array der Argumente ist in Java nicht mit `argv` in C und Unix identisch. Insbesondere `arg[0]` oder `arguments[0]`, das erste Element im Argumenten-Array, ist das erste Argument in der Befehlszeile nach dem Namen der Klasse, es ist *nicht* der Name des Programms wie in C. Achten Sie darauf, wenn Sie Ihre Java-Programme schreiben.

Nehmen wir beispielsweise an, Sie haben ein sehr einfaches Java-Programm namens `SumAverage`, das eine beliebige Anzahl an numerischen Argumenten erhält und die Summe sowie den Mittelwert dieser Argumente ausgibt. Eine erste Möglichkeit der Weitergabe an dieses Programm ist in Listing 6.6 aufgeführt. Probieren Sie nicht, diese Version zu kompilieren. Sehen Sie sich einfach den Code an, und versuchen Sie herauszufinden, was dieser tut.

Listing 6.6: Erster Versuch von `SumAverage.java`

```
1: class SumAverage {
2:     public static void main(String arguments[]) {
```

```

3:         int sum = 0;
4:
5:         for (int i = 0; i < arguments.length; i++) {
6:             sum += arguments[i];
7:         }
8:
9:         System.out.println("Sum is: " + sum);
10:        System.out.println("Average is: " +
11:            (float)sum / arguments.length);
12:    }
13: }

```

Auf den ersten Blick sieht dieses Programm ganz einfach und übersichtlich aus. Eine for-Schleife geht das Argumenten-Array durch, dabei werden die Summe und der Mittelwert ermittelt und als letzter Schritt ausgegeben.

Was passiert aber, wenn Sie versuchen, dieses Programm zu kompilieren? Sie erhalten folgenden Fehler:

```
SumAverage.java:9:Incompatible type for +=. Can't convert java.lang.String to int.
sum += args[i];
```

Diese Fehlermeldung erscheint, weil das Argumenten-Array ein String-Array ist. Obwohl Ganzzahlen an das Programm in der Befehlszeile weitergereicht wurden, werden diese Ganzzahlen in Strings konvertiert und dann im Array gespeichert. Um diese Ganzzahlen summieren zu können, müssen sie von Strings zurück in Ganzzahlen konvertiert werden. In der Klasse Integer gibt es die Klassenmethode `parseInt()`, die genau diesem Zweck dient. Wenn Sie Zeile 6 so abändern, daß hier diese Methode verwendet wird, funktioniert das Programm:

```
sum += Integer.parseInt(args[i]);
```

Wenn Sie jetzt das Programm kompilieren, läuft es fehlerfrei ab und gibt die erwarteten Ergebnisse aus. `java SumAverage 1 2 3` ergibt beispielsweise folgende Ausgabe:

```
Sum is: 6
Average is: 2
```

Zusammenfassung

Nach dem heutigen Tag sollten Sie wissen, warum Java Klasse hat. Alles, was Sie in Java erstellen, bezieht eine Hauptklasse ein, die mit anderen Klassen nach Bedarf interagiert. Dies stellt einen ganz anderen Ansatz dar, als es bei anderen Programmiersprachen der Fall ist.

Heute haben Sie alle Teile, die Sie in den bisherigen Lektionen dieser Woche gelernt haben, vereint, um Java-Klassen zu erstellen und sie in Java-Applikationen zu benutzen. Im einzelnen haben Sie in dieser Lektion folgendes gelernt:

- Instanz- und Klassenvariablen, die Attribute der Klasse und ihrer Instanzen darstellen. Sie haben gelernt, wie sie deklariert werden, wie sie sich von den üblichen lokalen Variablen unterscheiden und wie Konstanten deklariert werden.
- Instanz- und Klassenmethoden, die das Verhalten einer Klasse bestimmen. Sie haben gelernt, wie Methoden definiert werden, wie sich die Signatur einer Methode zusammensetzt, wie Methoden Werte ausgeben, wie Argumente zwischen Methoden weitergereicht werden und wie das Schlüsselwort `this` als Referenz auf das aktuelle Objekt angewandt werden kann.
- Java-Applikationen: Sie haben alles über die Methode `main()` gelernt, wie sie funktioniert und wie Argumente von einer Befehlszeile an eine Java-Applikation übergeben werden.

Morgen werden Sie die erste Woche abschließen und dabei einige fortgeschrittene Aspekte der Programmierung mit Methoden lernen.

Bis dahin hat die Klasse frei.

Fragen und Antworten

Frage:

Sie haben erwähnt, daß konstante, lokale Variablen in Applets, die zu Java 1.02 kompatibel sein sollen, nicht

erzeugt werden können. Warum sollte ich überhaupt Programme erstellen, die die aktuellen Features der Sprache in Java 1.2 nicht nutzen?

Antwort:

Der wahrscheinlichste Grund ist, daß Sie versuchen, ein Applet zu programmieren, das mit den meisten Browsern zusammenarbeitet. Die volle Unterstützung der Java-Versionen nach 1.02 fehlt noch in Browsern wie dem Netscape Navigator und dem Microsoft Internet Explorer, obwohl JavaSoft an Möglichkeiten arbeitet, dies zu beheben. Die gesamte Situation wird an Tag 8 besprochen.

Frage:

Eine meiner Klassen hat eine Instanzvariable namens origin. Ferner habe ich eine lokale Variable namens origin in einer Methode, die aufgrund des Gültigkeitsbereiches von der lokalen Variablen verborgen wird. Gibt es eine Möglichkeit, den Wert der Instanzvariablen zu erhalten?

Antwort:

Die einfachste Möglichkeit ist, die lokale Variable nicht genauso zu benennen wie die Instanzvariable. Falls Sie unbedingt den gleichen Namen verwenden wollen, können Sie this.origin verwenden, um spezifisch auf die Instanzvariable zu verweisen, während Sie als Referenz auf die lokale Variable origin verwenden.

Frage:

Ich habe ein Programm für vier Argumente geschrieben, wenn ich aber weniger Argumente angebe, stürzt das Programm ab, und ich erhalte einen Laufzeitfehler.

Antwort:

Prüfen Sie Zahl und Typ der Argumente, die Ihr Programm erwartet. Java überprüft das nicht automatisch für Sie. Verlangt Ihr Programm vier Argumente, müssen es tatsächlich vier sein, ansonsten erhalten Sie eine Fehlermeldung.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).

Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 1

Tag 7

Mehr über Methoden

Jede Klasse in Java agiert mit Methoden. Methoden sind wohl der wichtigste Teil einer objektorientierten Sprache, da sie jede Aktion, die ein Objekt ausführt, definieren.

Klassen und Objekte stellen ein Gerüst dar. Klassen- und Instanzvariablen bieten eine Möglichkeit zu beschreiben, was diese Klassen und Objekte sind. Allerdings können nur Methoden die Verhaltensweisen eines Objekts beschreiben - sprich die Dinge, die ein Objekt in der Lage ist zu tun, und die Art, mit der es mit anderen Klassen und Objekten kommuniziert.

Gestern haben Sie ein wenig über das Definieren von Methoden gelernt. Mit diesem Grundwissen können Sie bereits verschiedene Java-Programme schreiben, jedoch würden Ihnen einige Merkmale von Methoden fehlen, durch die Java-Programme erst richtig leistungstark werden. Durch sie werden Ihre Objekte und Klassen effizienter und übersichtlicher. Heute lernen Sie alles über diese zusätzlichen Merkmale, darunter:

- Überladen von Methoden, d.h. das Erstellen polymorpher Methoden - Methoden mit mehreren Signaturen und Definitionen, jedoch dem gleichen Namen
- Konstruktoren - Methoden, mit denen Sie Objekte initialisieren und beim Erstellen eines Objekts im System einen Anfangszustand einrichten können
- Überschreiben von Methoden - Erstellen einer neuen Definition für eine Methode, die in einer Superklasse definiert wurde, in einer Subklasse
- Finalizer-Methoden - Möglichkeiten für Objekte, sich selbst »aufzuräumen«, bevor sie aus dem Speicher entfernt werden

Methoden mit dem gleichen Namen, aber anderen Argumenten erstellen

Wenn Sie mit der Klassenbibliothek von Java arbeiten, werden Sie oft auf Klassen stoßen, die diverse Methoden mit demselben Namen besitzen. Die Klasse `java.lang.String` z.B. verfügt über einige verschiedene `valueOf()`-Methoden.

Methoden mit demselben Namen werden durch zwei Dinge voneinander unterschieden:

- Die Anzahl der Argumente, die ihnen übergeben wird
- Den Datentyp oder Objekttyp der einzelnen Argumente

Diese beiden Dinge bilden die Signatur einer Methode. Mehrere Methoden zu verwenden, die alle denselben Namen, aber unterschiedliche Signaturen haben, wird als überladen (engl. *overloading*) bezeichnet.

In dem Beispiel der Klasse `String` von oben verarbeiten die verschiedenen überladenen Versionen der `valueOf()`-Methode unterschiedliche Datentypen als Parameter.

Durch die Überladung von Methoden besteht kein Bedarf mehr für völlig verschiedene Methoden, die im wesentlichen dasselbe tun. Über die Überladung wird es auch möglich, daß sich Methoden in Abhängigkeit von den erhaltenen Argumenten unterschiedlich verhalten.

Die überladene Methode `valueOf()` in der Klasse `String` kann zur Konvertierung einer ganzen Reihe unterschiedlicher Datentypen und Objekte in `String`-Werte verwendet werden. Wenn Sie eine Methode in einem Objekt aufrufen, überprüft Java den Methodennamen und die Argumente, um zu ermitteln, welche Methodendefinition auszuführen ist.

Um eine Methode zu überladen, legen Sie lediglich mehrere unterschiedliche Methodendefinitionen in einer Klasse an, die alle den gleichen Namen, jedoch unterschiedliche Parameter (entweder in Bezug auf die Zahl oder den Typ der Argumente) und unterschiedliche Rümpfe haben. Java erkennt überladene Methoden daran, daß die einzelnen Parameterlisten für jeden Methodennamen eindeutig sind.

Java unterscheidet überladene Methoden mit dem gleichen Namen anhand der Zahl und des Typs der Parameter für die jeweilige Methode, nicht anhand des Rückgabetyps. Das heißt, wenn Sie zwei Methoden mit dem gleichen Namen, der gleichen Parameterliste, jedoch unterschiedlichen Rückgabetypen erstellen, erhalten Sie einen Compilerfehler. Die für

jeden Parameter einer Methode gewählten Variablennamen sind nicht relevant, nur die Zahl und der Typ zählen.

Im folgenden Beispiel wird eine überladene Methode erstellt. Listing 7.1 zeigt eine einfache Klassendefinition für die Klasse `MyRect`, die ein Rechteck definiert. Die Klasse `MyRect` hat vier Instanzvariablen, die die obere linke und untere rechte Ecke des Rechtecks definieren: `x1`, `y1`, `x2` und `y2`.



Warum habe ich die Klasse `MyRect` genannt? Im `awt`-Paket von Java ist eine Klasse namens `Rectangle` enthalten, die einen Großteil des gleichen Verhaltens implementiert. Um zu vermeiden, daß die zwei Klassen verwechselt werden, habe ich diese Klasse `MyRect` genannt.

Listing 7.1: Die Anfänge der Klasse `MyRect`

```
1: class MyRect {
2:     int x1 = 0;
3:     int y1 = 0;
4:     int x2 = 0;
5:     int y2 = 0;
6: }
```



Versuchen Sie nicht, dieses Beispiel an dieser Stelle bereits zu kompilieren. Es wird zwar kompiliert werden, ohne Fehler zu erzeugen, allerdings wird es nicht laufen, da es (noch) keine `main()`-Methode besitzt. Wenn die Klassendefinition fertig ist, wird die endgültige Version sich sowohl kompilieren als auch ausführen lassen.

Wird eine neue Instanz von der Klasse `MyRect` erstellt, werden alle ihre Instanzvariablen mit 0 initialisiert. Wir definieren nun die Methode `buildRect()`, die die Größe des Rechtecks anhand von vier Integer-Argumenten auf die entsprechenden Werte abändert, so daß das Rechteckobjekt richtig ausgegeben wird (da die Argumente den gleichen Namen als Instanzvariablen haben, müssen Sie sicherstellen, daß auf sie mit `this` verwiesen wird):

```
MyRect buildRect (int x1, int y1, int x2, int y2) {
this.x1 = x1;
this.y1 = y1;
this.x2 = x2;
this.y2 = y2;
return this;
}
```

Was, wenn man nun die Dimensionen des Rechtecks auf andere Weise definieren will, z.B. durch Verwendung von `Point`-Objekten anstelle der einzelnen Koordinaten? In diesem Fall überladen Sie `buildRect()`, so daß dessen Parameterliste zwei `Point`-Objekte erhält:

```
MyRect buildRect (Point topLeft, Point bottomRight) {
x1 = topLeft.x;
y1 = topLeft.y;
x2 = bottomRight.x;
y2 = bottomRight.y;
return this;
}
```

Damit die obige Methode funktioniert, müssen Sie die `Point`-Klasse an den Anfang Ihrer Quelldatei importieren, so daß Java sie finden kann.

Eventuell möchten Sie das Rechteck mit einer oberen Ecke sowie einer bestimmten Breite und Höhe definieren. Hierfür schreiben Sie einfach eine andere Definition für `buildRect()`:

```
MyRect buildRect (Point topLeft, int w, int h) {
x1 = topLeft.x;
y1 = topLeft.y;
```



```

x2 = (x1 + w);
y2 = (y1 + h);
return this;
}

```

Um dieses Beispiel zu beenden, erstellen wir eine Methode `printRect()` zum Ausgeben der Koordinaten des Rechtecks und eine `main()`-Methode zum Testen aller Werte (einfach um zu beweisen, daß das tatsächlich funktioniert). Listing 7.2 zeigt die fertige Klassendefinition mit allen Methoden: drei `buildRect()`-Methoden, eine `printRect()`-Methode und eine `main()`-Methode.

Listing 7.2: Der komplette Quelltext von `MyRect.java`

```

1: import java.awt.Point;
2:
3: class MyRect {
4:     int x1 = 0;
5:     int y1 = 0;
6:     int x2 = 0;
7:     int y2 = 0;
8:
9:     MyRect buildRect(int x1, int y1, int x2, int y2) {
10:         this.x1 = x1;
11:         this.y1 = y1;
12:         this.x2 = x2;
13:         this.y2 = y2;
14:         return this;
15:     }
16:
17:     MyRect buildRect(Point topLeft, Point bottomRight) {
18:         x1 = topLeft.x;
19:         y1 = topLeft.y;
20:         x2 = bottomRight.x;
21:         y2 = bottomRight.y;
22:         return this;
23:     }
24:
25:     MyRect buildRect(Point topLeft, int w, int h) {
26:         x1 = topLeft.x;
27:         y1 = topLeft.y;
28:         x2 = (x1 + w);
29:         y2 = (y1 + h);
30:         return this;
31:     }
32:
33:     void printRect(){
34:         System.out.print("MyRect: <" + x1 + ", " + y1);
35:         System.out.println(", " + x2 + ", " + y2 + ">");
36:     }
37:
38:     public static void main(String arguments[]) {
39:         MyRect rect = new MyRect();
40:
41:         System.out.println("Calling buildRect with coordinates 25,25,
42:         & 50,50:");
43:         rect.buildRect(25, 25, 50, 50);
44:         rect.printRect();
45:         System.out.println("***");
46:
47:         System.out.println("Calling buildRect with points (10,10), (20,20):");
48:         rect.buildRect(new Point(10,10), new Point(20,20));

```

```

48:         rect.printRect();
49:         System.out.println("***");
50:
51:         System.out.print("Calling buildRect with 1 point (10,10),");
52:         System.out.println(" width (50) and height (50):");
53:
54:         rect.buildRect(new Point(10,10), 50, 50);
55:         rect.printRect();
56:         System.out.println("***");
57:     }
58: }

```

Die Ausgabe dieses Java-Programms ist wie folgt:

Calling buildRect with coordinates 25,25, 50,50:

MyRect: <25, 25, 50, 50>

Calling buildRect with points (10,10), (20,20):

MyRect: <10, 10, 20, 20>

Calling buildRect with 1 point (10,10), width (50) and height (50):

MyRect: <10, 10, 60, 60>

Wie Sie an diesem Beispiel sehen, funktionieren alle buildRect()-Methoden auf der Grundlage der Argumente, mit denen sie aufgerufen werden. Sie können in Ihren Klassen beliebig viele Versionen einer Methode definieren, um das für die jeweilige Klasse benötigte Verhalten zu implementieren.

Konstruktor-Methoden

Zusätzlich zu den üblichen Methoden können Sie in einer Klassendefinition auch Konstruktor-Methoden definieren.



Eine Konstruktor-Methode oder auch nur Konstruktor ist eine besondere Methodenart, die beim Erstellen eines Objekts aufgerufen wird - mit anderen Worten, wenn ein Objekt konstruiert wird.

Im Gegensatz zu den üblichen Methoden können Sie einen Konstruktor nicht direkt aufrufen. Konstruktoren werden statt dessen von Java automatisch aufgerufen.

Wenn mit new eine neue Klasseninstanz erstellt wird, führt Java drei Aktionen aus:

- Speicherzuweisung für das Objekt
- Initialisierung der Instanzvariablen des Objekts auf ihre Anfangswerte oder auf einen Standard (0 bei Zahlen, null bei Objekten und false bei booleschen Werten oder '\0' bei Zeichen)
- Aufruf des Konstruktors der Klasse (die eine von mehreren Methoden sein kann)

Auch wenn für eine Klasse keine speziellen Konstruktoren definiert wurden, erhalten Sie trotzdem ein Objekt, müssen aber seine Instanzvariablen setzen oder andere Methoden aufrufen, die das Objekt zur Initialisierung braucht. Alle Beispiele, die Sie bisher geschrieben haben, verhalten sich so.

Durch Definieren von Konstruktoren in Ihren Klassen können Sie Anfangswerte von Instanzvariablen setzen, Methoden anhand dieser Variablen oder Methoden für andere Objekte aufrufen und die anfänglichen Eigenschaften Ihres Objekts bestimmen. Ferner können Sie Konstruktoren überladen wie übliche Methoden, um ein Objekt zu erstellen, das die spezifischen Merkmale entsprechend den in new festgelegten Argumenten aufweist.

Basis-Konstruktoren

Konstruktoren sehen wie übliche Methoden aus, unterscheiden sich von diesen aber durch zwei Merkmale:

- Konstruktoren haben immer den gleichen Namen wie die Klasse.

- Konstruktoren haben keinen Rückgabetyt.

Listing 7.3 ist ein Beispiel mit einer einfachen Klasse namens Person. Diese Klasse verfügt über einen Konstruktor, der die Instanzvariablen der Klasse anhand der Argumente von new initialisiert. Ferner beinhaltet die Klasse eine Methode für das Objekt, damit es sich selbst vorstellen kann.

Listing 7.3: Die Klasse Person

```

1: class Person {
2:     String name;
3:     int age;
4:
5:     Person(String n, int a) {
6:         name = n;
7:         age = a;
8:     }
9:
10:    void printPerson() {
11:        System.out.print("Hi, my name is " + name);
12:        System.out.println(". I am " + age + " years old.");
13:    }
14:
15:    public static void main (String args[]) {
16:        Person p;
17:        p = new Person("Luke", 50);
18:        p.printPerson();
19:        System.out.println("----");
20:        p = new Person("Laura", 35);
21:        p.printPerson();
22:        System.out.println("----");
23:    }
24: }
```

Die Ausgabe dieses Beispiels ist wie folgt:

```

Hi, my name is Luke. I am 50 years old.
----
Hi, my name is Laura. I am 35 years old.
----
```

Die Klasse Person besitzt drei Methoden: Die erste ist der Konstruktor, der in den Zeilen 5 bis 8 definiert wird. Dieser initialisiert die beiden Instanzvariablen der Klasse mit den Werten der Argumente, die beim Erstellen einer Instanz der Klasse mit new übergeben werden. Die Klasse Person verfügt außerdem über die Methode printPerson() , so daß das Objekt sich selbst vorstellen kann. Schließlich gibt es noch eine main()-Methode, um die einzelnen Teile zu testen.

Aufrufen eines anderen Konstruktors

In manchen Fällen werden Sie eine Konstruktor-Methode definieren, die das Verhalten einer bereits vorhandenen Konstruktor-Methode dupliziert und gleichzeitig um bestimmte neue Verhaltensweisen erweitert. Anstatt nun die identischen Verhaltensweisen in diverse Konstruktor-Methoden in Ihrer Klasse zu kopieren, können Sie die erste Konstruktor-Methode aus dem Rumpf der anderen aufrufen. Java stellt hierfür eine spezielle Syntax zur Verfügung. Sie verwenden folgende Form, um einen Konstruktor, der in der aktuellen Klasse definiert ist, aufzurufen:

```
this(arg1, arg2, arg3...);
```

Die Verwendung von this in bezug auf eine Konstruktor-Methode ist ganz ähnlich wie beim Zugriff auf die Variablen des aktuellen Objekts mit this. Die für this() verwendeten Argumente sind selbstverständlich die Argumente des Konstruktors. Der Konstruktor der Klasse Person aus Listing 7.2 könnte z.B. wie folgt aus einem anderen Konstruktor heraus aufgerufen werden:

```
this(n, a);
```

n ist dabei der String, der den Namen des Person-Objekts enthält, und a der Integer, der das Alter angibt.

Konstruktoren überladen

Wie die üblichen Methoden können auch Konstruktoren überladen werden, also verschiedene Anzahl und Typen von Parametern annehmen. Dies gibt Ihnen die Möglichkeit, Objekte mit genau den gewünschten Eigenschaften zu erstellen, und Sie sind in der Lage, Eigenschaften aus verschiedenen Eingaben zu berechnen.

Beispielsweise sind die Methoden `buildRect()`, die Sie heute in der `MyRect`-Klasse definiert haben, ausgezeichnete Konstruktoren, weil sie die Instanzvariablen eines Objekts auf die entsprechenden Werte initialisieren. Das bedeutet, daß Sie anstelle der ursprünglich definierten Methode `buildRect()` (die vier Parameter für die Eckkoordinaten heranzieht) einen Konstruktor erstellen können. In Listing 7.4 wird die neue Klasse `MyRect2` aufgeführt, die die gleiche Funktionalität aufweist wie die ursprüngliche Klasse `MyRect`. Jedoch wurde sie anstelle der Methode `buildRect()` mit überladenen Konstruktor-Methoden angelegt.

Listing 7.4: Der gesamte Quelltext von `MyRect2.java`

```

1: import java.awt.Point;
2:
3: class MyRect2 {
4:     int x1 = 0;
5:     int y1 = 0;
6:     int x2 = 0;
7:     int y2 = 0;
8:
9:     MyRect2(int x1, int y1, int x2, int y2) {
10:         this.x1 = x1;
11:         this.y1 = y1;
12:         this.x2 = x2;
13:         this.y2 = y2;
14:     }
15:
16:     MyRect2(Point topLeft, Point bottomRight) {
17:         x1 = topLeft.x;
18:         y1 = topLeft.y;
19:         x2 = bottomRight.x;
20:         y2 = bottomRight.y;
21:     }
22:
23:     MyRect2(Point topLeft, int w, int h) {
24:         x1 = topLeft.x;
25:         y1 = topLeft.y;
26:         x2 = (x1 + w);
27:         y2 = (y1 + h);
28:     }
29:
30:     void printRect() {
31:         System.out.print("MyRect: <" + x1 + ", " + y1);
32:         System.out.println(", " + x2 + ", " + y2 + ">");
33:     }
34:
35:     public static void main(String arguments[]) {
36:         MyRect2 rect;
37:
38:         System.out.println("Calling MyRect2 with coordinates 25,25 50,50:");
39:         rect = new MyRect2(25, 25, 50,50);
40:         rect.printRect();
41:         System.out.println("***");
42:
43:         System.out.println("Calling MyRect2 with points (10,10), (20,20):");
44:         rect= new MyRect2(new Point(10,10), new Point(20,20));
45:         rect.printRect();

```

```

46:         System.out.println("***");
47:
48:         System.out.print("Calling MyRect2 with 1 point (10,10)");
49:         System.out.println(" width (50) and height (50):");
50:         rect = new MyRect2(new Point(10,10), 50, 50);
51:         rect.printRect();
52:         System.out.println("***");
53:
54:     }
55: }

```

Nachfolgend sehen Sie die von diesem Beispielprogramm produzierte Ausgabe (sie ist mit der des vorherigen Beispiels identisch, nur der dafür nötige Code wurde geändert):

```

Calling MyRect2 with coordinates 25,25 50,50:
MyRect: <25, 25, 50, 50>
***

```

```

Calling MyRect2 with points (10,10), (20,20):
MyRect: <10, 10, 20, 20>
***

```

```

Calling MyRect2 with 1 point (10,10) width (50) and height (50):
MyRect: <10, 10, 60, 60>
***

```

Methoden überschreiben

Wenn Sie eine Methode in einem Objekt aufrufen, sucht Java nach der Methodendefinition in der Klasse dieses Objekts. Falls es keine übereinstimmende Signatur findet, wird der Methodenaufruf in der Klassenhierarchie nach oben weitergereicht, bis eine passende Methodendefinition gefunden wird. Durch die in Java implementierte Methodenvererbung können Sie Methoden wiederholt in Subklassen definieren und verwenden, ohne den Code an sich duplizieren zu müssen.

Zuweilen soll ein Objekt aber auf die gleichen Methoden reagieren, jedoch beim Aufrufen der jeweiligen Methode ein anderes Verhalten aufweisen. In diesem Fall können Sie die Methode überschreiben. Durch Überschreiben von Methoden definieren Sie eine Methode in einer Subklasse, die die gleiche Signatur hat wie eine Methode in einer Superklasse. Dann wird zum Zeitpunkt des Aufrufs nicht die Methode in der Superklasse, sondern die in der Subklasse ermittelt und ausgeführt.

Erstellen von Methoden, die andere überschreiben

Um eine Methode zu überschreiben, erstellen Sie eine Methode in der Subklasse, die die gleiche Signatur (Name, Rückgabetyp und Parameterliste) hat wie eine Methode, die in einer Superklasse der betreffenden Klasse definiert wurde. Da Java die erste Methodendefinition ausführt, die es findet und die mit der Signatur übereinstimmt, wird die ursprüngliche Methodendefinition dadurch »verborgen«. Wir betrachten im folgenden ein einfaches Beispiel. Listing 7.5 zeigt eine einfache Klasse mit der Methode `printMe()`, die den Namen der Klasse und die Werte ihrer Instanzvariablen ausgibt.

Listing 7.5: Der Quelltext von `PrintClass.java`

```

1: class PrintClass {
2:     int x = 0;
3:     int y = 1;
4:
5:     void printMe() {
6:         System.out.println("x is " + x + ", y is " + y);
7:         System.out.println("I am an instance of the class " +
8:             this.getClass().getName());
9:     }
10: }

```

Listing 7.6 umfaßt eine Klasse namens `PrintSubClass`, die eine Subklasse von `PrintClass` ist. Der einzige Unterschied

zwischen PrintClass und PrintSubClass ist, daß letztere die Instanzvariable z hat.

Listing 7.6: Der Quelltext von PrintSubClass.java

```

1: class PrintSubClass extends PrintClass {
2:     int z = 3;
3:
4:     public static void main(String args[]) {
5:         PrintSubClass obj = new PrintSubClass();
6:         obj.printMe();
7:     }
8: }
```

Im folgenden die Ausgabe von PrintSubClass:

```

X is 0, Y is 1
I am an instance of the class PrintSubClass
```

In der Methode main() von PrintSubClass erstellen Sie ein PrintSubClass-Objekt und rufen die Methode printMe() auf. Beachten Sie, daß PrintSubClass diese Methode nicht definiert; deshalb sucht Java in allen Superklassen von PrintSubClass danach und findet sie in diesem Fall in PrintClass. Leider wird die Instanzvariable z nicht ausgegeben, wie Sie in der Ausgabe oben sehen können.



Die Klasse PrintClass besitzt ein wichtiges Feature, auf das hier eingegangen werden soll: Sie hat keine main()-Methode. Sie braucht auch keine, da sie keine Applikation ist. PrintClass ist lediglich eine Hilfsklasse für die Klasse PrintSubClass, die eine Applikation ist und deshalb auch über eine main()-Methode verfügt. Nur die Klasse, die Sie mit dem Java-Interpreter ausführen, benötigt eine main()-Methode.

Wir erstellen nun eine dritte Klasse. PrintSubClass2 ist fast mit PrintSubClass identisch, jedoch wird die Methode printMe() überschrieben, um die Variable z einzubinden. Listing 7.7 zeigt diese Klasse.

Listing 7.7: Die PrintSubClass2-Klasse

```

1: class PrintSubClass2 extends PrintClass {
2:     int z = 3;
3:
4:     void printMe() {
5:         System.out.println("x is " + x + ", y is " + y +
6:             ", z is " + z);
7:         System.out.println("I am an instance of the class " +
8:             this.getClass().getName());
9:     }
10:
11:     public static void main(String args[]) {
12:         PrintSubClass2 obj = new PrintSubClass2();
13:         obj.printMe();
14:     }
15: }
```

Wenn Sie diese Klasse instanzieren und die Methode printMe() aufrufen, wird diesmal die Version von printMe(), die Sie für diese Klasse definiert haben, und nicht diejenige, die sich in der Superklasse PrintClass befindet, aufgerufen.

Das sehen Sie an folgender Ausgabe:

```

x is 0, y is 1, z is 3
I am an instance of the class PrintSubClass2
```

Aufrufen der Originalmethode

Normalerweise gibt es zwei Gründe dafür, warum man eine Methode, die in einer Superklasse bereits implementiert ist, überschreiben will:

- Um die Definition der Originalmethode völlig zu ersetzen
- Um die Originalmethode zu erweitern

Den ersten Grund haben Sie bereits kennengelernt. Durch Überladen einer Methode und Schreiben einer neuen Definition für diese Methode haben Sie die ursprüngliche Definition der Methode verborgen. Zuweilen will man aber auch die ursprüngliche Definition nicht ganz ersetzen, sondern vielmehr um zusätzliche Eigenschaften erweitern. Das ist besonders nützlich, wenn es darauf hinausläuft, Verhaltensweisen der Originalmethode in der neuen Definition zu duplizieren. Sie können die Originalmethode im Rumpf der überschriebenen Methode aufrufen und alles hinzufügen, was Sie benötigen.

Um die Originalmethode innerhalb einer Methodendefinition aufzurufen, benutzen Sie das Schlüsselwort `super`, um den Methodenaufruf in der Hierarchie nach oben weiterzugeben:

```
void myMethod (String a, String b) {
// Hier irgendwas ausführen
super.myMethod(a, b);
// Eventuell weitere Anweisungen
}
```

Wie `this` ist auch das Schlüsselwort `super` ein Platzhalter - in diesem Fall für die Superklasse dieser Klasse. Sie können es überall dort verwenden, wo Sie auch `this` verwenden. `super` verweist allerdings auf die Superklasse und nicht auf die aktuelle Klasse.

In Listing 7.8 sehen Sie die `printMe()`-Methoden aus dem vorherigen Beispiel.

Listing 7.8: Die `printMe`-Methoden

```
1: // aus PrintClass
2: void printMe() {
3:     System.out.println("x is " + x + ", y is " + y);
4:     System.out.println("I am an instance of the class" +
5:         this.getClass().getName());
6: }
7: }
8:
9: // aus PrintSubClass2
10: void printMe() {
11:     System.out.println("x is " + x + ", y is " + y + ", z is " + z);
12:     System.out.println("I am an instance of the class " +
13:         this.getClass().getName());
14: }
```

Anstatt den Großteil des Verhaltens von der Methode der Superklasse in die Subklasse zu duplizieren, können Sie die Methode der Superklasse so umstellen, daß zusätzliche Eigenschaften mühelos hinzugefügt werden können:

```
// von PrintClass
void printMe() {
System.out.println("I am an instance of the class" +
this.getClass().getName());
System.out.println("X is " + x);
System.out.println("Y is " + y);
}
```

Beim Überschreiben von `printMe()` können Sie dann in der Superklasse die Originalmethode aufrufen und alles hinzufügen, was Sie benötigen:

```
// von PrintSubClass2
void printMe() {
super.printMe();
System.out.println("Z is " + z);
}
```

```
}

```

Die Ausgabe des Aufrufs von `printMe()` in einer Instanz der Superklasse sieht so aus:

```
I am an instance of the class PrintSubClass2
X is 0
Y is 1
Z is 3

```

Konstruktoren überschreiben

Konstruktoren können technisch nicht überschrieben werden. Da sie immer den gleichen Namen haben wie die aktuelle Klasse, werden Konstruktoren nicht vererbt, sondern immer neu erstellt. Das ist in den meisten Fällen sinnvoll, denn wenn ein Konstruktor Ihrer Klasse aufgerufen wird, wird gleichzeitig auch der Konstruktor mit der gleichen Signatur aller Superklassen aktiviert, so daß eventuell alle Teile einer Klasse initialisiert werden.

Andererseits möchten Sie eventuell beim Definieren von Konstruktoren für eine Klasse einiges ändern, z.B. wie ein Objekt initialisiert wird. Eventuell soll es nicht durch Initialisieren der Informationen, die Ihre Klasse zusätzlich einbringt, sondern durch Änderung der bereits vorhandenen Informationen initialisiert werden. Sie erreichen das, indem Sie die Konstruktoren Ihrer Superklasse explizit aufrufen.

Um eine normale Methode in einer Superklasse aufzurufen, benutzen Sie `super.methodname(argumente)`. Da Sie bei Konstruktoren keinen Methodennamen aufrufen müssen, wenden Sie hier eine andere Form an:

```
super(arg1, arg2, ...);

```

Beachten Sie allerdings, daß es in Java eine ganz spezielle Regel für die Verwendung von `super()` gibt: Es muß die allererste Anweisung in der Definition Ihres Konstruktors sein. Wenn Sie `super()` nicht explizit in Ihrem Konstruktor aufrufen, dann erledigt Java dies für Sie und verwendet dafür `super()` ohne Argumente.

Ebenso wie bei der Verwendung von `this(...)` bewirkt `super(...)` in Konstruktor-Methoden den Aufruf der Konstruktor-Methode für die unmittelbare Superklasse (die ihrerseits den Konstruktor ihrer Superklasse aufrufen kann usw.). Beachten Sie bitte, daß in der Superklasse ein Konstruktor mit der entsprechenden Signatur vorhanden sein muß, damit der Aufruf von `super()` funktioniert. Der Java-Compiler überprüft dies, wenn er versucht, die Quelldatei zu kompilieren.

Den Konstruktor in der Superklasse Ihrer Klasse mit derselben Signatur müssen Sie nicht extra aufrufen. Sie müssen lediglich den Konstruktor für die Werte aufrufen, die initialisiert werden müssen. Sie können sogar eine Klasse erstellen, die über Konstruktoren mit total anderen Signaturen als die Konstruktoren in einer der Superklassen verfügt.

Das Beispiel in Listing 7.9 zeigt die Klasse `NamedPoint`, die sich von der Klasse `Point` aus dem `awt`-Paket von Java ableitet. Die `Point`-Klasse hat nur einen Konstruktor, der die Argumente `x` und `y` entgegennimmt und ein `Point`-Objekt zurückgibt. `NamedPoint` hat eine zusätzliche Instanzvariable (einen `String` für den Namen) und definiert einen Konstruktor, um `x`, `y` und den Namen zu initialisieren.

Listing 7.9: Die `NamedPoint`-Klasse

```
1: import java.awt.Point;
2: class NamedPoint extends Point {
3:     String name;
4:
5:     NamedPoint(int x, int y, String name) {
6:         super(x,y);
7:         this.name = name;
8:     }
9:     public static void main (String arg[]) {
10:         NamedPoint np = new NamedPoint(5, 5, "SmallPoint");
11:         System.out.println("x is " + np.x);
12:         System.out.println("y is " + np.y);
13:         System.out.println("Name is " + np.name);
14:     }
15: }
```

Das Programm liefert die folgende Ausgabe:

```
x is 5
y is 5
Name is SmallPoint
```

Der hier für NamedPoint definierte Konstruktor ruft die Konstruktor-Methode von Point auf, um die Instanzvariablen (x und y) von Point zu initialisieren. Obwohl Sie x und y ebenso gut selbst initialisieren könnten, wissen Sie eventuell nicht, ob Point noch andere Operationen ausführt, um sich zu initialisieren. Deshalb ist es immer ratsam, Konstruktor-Methoden nach oben in der Hierarchie weiterzugeben, um sicherzustellen, daß alles richtig gesetzt wird.

Finalizer-Methoden

Finalizer-Methoden sind in gewissem Sinn das Gegenstück zu Konstruktor-Methoden. Während eine Konstruktor-Methode benutzt wird, um ein Objekt zu initialisieren, werden Finalizer-Methoden aufgerufen, kurz bevor das Objekt im Papierkorb landet und sein Speicher zurückgefordert wird.

Die Finalizer-Methode hat den Namen finalize(). Die Klasse Object definiert eine Standard-Finalizer-Methode, die keine Funktionalität hat. Um eine Finalizer-Methode zu erstellen, tragen Sie eine Methode mit der folgenden Signatur in Ihre Klassendefinition ein:

```
protected void finalize() throws Throwable{
super.finalize
}
```



Der Teil throws Throwable dieser Methodendefinition bezieht sich auf die Fehler, die nach dem Aufruf dieser Methode eventuell auftreten. Fehler werden in Java Exceptions oder zu deutsch Ausnahmen genannt. Am Tag 17 lernen Sie mehr darüber. Vorerst reicht es, daß Sie diese Schlüsselwörter in die Methodendefinition aufnehmen.

Im Körper dieser finalize()-Methode können Sie alle möglichen »Aufräumprozeduren« einbinden, die das Objekt ausführen soll. Sie können super.finalize() aufrufen, um die Aufräumarbeiten, wenn nötig, an die Superklasse Ihrer Klasse zu delegieren. (Dieses ist im allgemeinen recht sinnvoll, um es allen Beteiligten zu ermöglichen, das Objekt zu bearbeiten, wenn dies notwendig sein sollte.)

Sie können die Methode finalize() jederzeit auch selbst aufrufen. Es handelt sich um eine einfache Methode wie alle anderen. Allerdings sorgt der Aufruf von finalize() nicht dafür, daß der Garbage Collector für dieses Objekt aktiv wird. Nur durch Entfernen aller Referenzen auf das Objekt wird das Objekt zum Löschen markiert.

Finalizer-Methoden eignen sich am besten zur Optimierung des Entferns von Objekten, z.B. durch Löschen aller Referenzen auf andere Objekte. In den meisten Fällen benötigen Sie finalize() überhaupt nicht.

Zusammenfassung

Heute haben Sie verschiedene Techniken zum Benutzen, Wiederverwenden, Definieren und Neudefinieren von Methoden kennengelernt. Sie haben gelernt, wie man durch Überladen eines Methodennamens die gleiche Methode mit einem anderen Verhalten auf der Grundlage der Argumente, durch die sie aufgerufen wird, definiert. Sie haben viel über Konstruktor-Methoden gelernt, die benutzt werden, um ein neues Objekt beim Erstellen zu initialisieren. Sie haben mehr über Methodenvererbung erfahren und gelernt, wie Methoden in Subklassen einer Klasse durch Überschreiben definiert werden können. Außerdem haben Sie gelernt, daß Sie mit Finalizer-Methoden Ihr Programm »aufräumen« können.

Nachdem Sie sich nun einen Tag lang mit Javas Art, mit Methoden umzugehen, beschäftigt haben, sollten Sie bereit sein, Ihre eigenen Programme zu schreiben.

Beginnend mit der nächsten Woche werden Sie ausgefeiltere Programme schreiben, wobei Sie die Techniken von Java 1.02 für Applets und die von Java 1.2 für Applikationen verwenden. Sie werden mit Grafik, grafischen Benutzeroberflächen, Maus- und Tastaturereignissen und Fenstern arbeiten. Dies kann der Einstieg in eine große Zeit sein.

Fragen und Antworten

Frage:

Ich habe zwei Methoden mit folgenden Signaturen geschrieben:

```
int total(int arg1, int arg2, int arg3) { }  
float total(int arg1, int arg2, int arg3) {...}
```

Der Java-Compiler meckert beim Kompilieren der Klasse mit diesen Methodendefinitionen. Die Signaturen sind doch unterschiedlich. Wo liegt der Fehler?

Antwort:

Das Überladen von Methoden funktioniert in Java nur, wenn sich die Parameterlisten unterscheiden, entweder in der Zahl oder im Typ der Argumente. Die Rückgabetypen sind beim Überladen von Methoden nicht relevant. Wie soll Java bei zwei Methoden mit genau der gleichen Parameterliste wissen, welche aufzurufen ist?

Frage:

Kann ich auch Methoden, die bereits überschrieben wurden, überladen (d.h. kann ich Methoden erstellen, die den gleichen Namen wie eine geerbte Methode, jedoch eine andere Parameterliste haben)?

Antwort:

Sicher! Solange die Parameterliste unterschiedlich ist, spielt es keine Rolle, ob Sie einen neuen Methodennamen oder einen, den Sie aus einer Superklasse geerbt haben, definieren.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 2**Tag 8****Grundlagen der Java-Applets**

Seit seiner Einführung war Java der Rock Star der Computersprachen. Es hat eine öffentliche Aufmerksamkeit erfahren, die eigentlich den Skandalen von Präsidenten, Zuckerersatzstoffen und Profisportlern, die mit kriminellen Delikten zu tun haben, vorbehalten ist.

Die wesentliche Ursache für diesen Hype waren Applets: Java-Programme, die über das World Wide Web ausgeführt werden. Die meisten Leute hatten Ihre erste Begegnung mit Java Ende 1995, als der Netscape Navigator die Ausführung von Applets zu unterstützen begann.

Obwohl Java heute für viele Dinge außerhalb des Web verwendet werden kann, lernt eine nicht unerhebliche Zahl von Programmierern die Sprache, um Applets zu schreiben.

Letzte Woche haben Sie sich auf die Sprache selbst konzentriert, und bei den meisten kleinen Programmen, die Sie erstellt haben, handelte es sich um Java-Applikationen. Diese Woche schreiten Sie fort zur Entwicklung von Applets.

Heute beginnen wir mit den Grundlagen:

- Kleine Übersicht zu den Unterschieden zwischen Java-Applets und -Applikationen.
- So werden einfache Applets erstellt.
- So wird ein Applet auf einer Webseite eingefügt.
- So werden Informationen von einer Webseite zu einem Applet geschickt.
- So werden Applets und die zugehörigen Dateien in einem komprimierten Archiv gespeichert, um schnellere Ladezeiten zu erzielen.

Unterschiede zwischen Applets und Anwendungen

Der Unterschied zwischen Java-Applets und -Applikationen liegt in der Art, wie diese ausgeführt werden.

Applikationen werden mit dem Java-Interpreter ausgeführt, der die Haupt-.class-Datei der Applikation lädt. Dazu wird meist das java-Tool des JDK von der Kommandozeile aus aufgerufen, wie Sie das bereits am ersten Tag getan haben.

Java-Applets hingegen werden innerhalb eines WWW-Browsers ausgeführt, der Java unterstützt. Momentan schließt dies die aktuellen Versionen des Netscape Navigator, des Microsoft Internet Explorer und Suns HotJava ein. Applets können zusätzlich mit dem Applet-Viewer, der im Java Developer's Kit enthalten ist, angezeigt werden.

Um ein Applet auszuführen, muß es in eine Webseite eingefügt werden. Dies geschieht mit Hilfe von HTML-Tags, wie das auch bei Bildern und anderen Elementen der Fall ist. Wenn ein Anwender mit einem Java-fähigen Browser eine Webseite lädt, die ein Applet enthält, lädt der Browser das Applet von einem Web-Server herunter und führt es auf dem System des Benutzers aus. Es wird kein separater Java-Interpreter benötigt, da bereits einer in den Browser integriert ist. Wie eine Applikation hat auch ein Applet eine Haupt-.class-Datei und beliebige weitere .class-Dateien, die zur Ausführung des Applets benötigt werden. Die Standard-Klassenbibliothek von Java ist automatisch enthalten.

Da Java-Applets innerhalb eines Java-Browsers ausgeführt werden, ist ein Teil der Arbeit für die Erstellung einer Benutzeroberfläche bereits für den Applet-Programmierer getan. Es gibt ein Fenster, in dem das Applet ausgeführt werden kann, einen Bereich, in dem Grafiken angezeigt und Informationen empfangen werden können, sowie die

Benutzerschnittstelle des Browsers.



Es ist machbar, daß ein einziges Java-Programm sowohl als Java-Applikation ausgeführt werden kann als auch als Java-Applet. Obwohl Sie zum Erstellen von Applets und Anwendungen verschiedene Vorgehensweisen verwenden, stehen diese nicht in Konflikt miteinander. Die für Applets spezifischen Features werden ignoriert, sobald das Programm als Applikation ausgeführt wird und umgekehrt.

Sicherheitseinschränkungen von Applets

Da Java-Applets von jeder beliebigen Site im World Wide Web heruntergeladen werden können und auf dem System des Client ausgeführt werden, sind bestimmte Sicherheitsvorkehrungen getroffen worden. Ein boshafter Java-Programmierer könnte leicht ein Applet schreiben, das Dateien des Anwenders löscht, private Informationen auf dem System sammelt und andere Sicherheitsbrüche vornimmt.

In der Regel werden Java-Applets nach dem Sicherheitsmodell »Lieber sicher, als daß es einem später leid tut« ausgeführt. Die Dinge, die in der folgenden Liste aufgeführt sind, kann ein Applet nicht tun:

- Applets können im Dateisystem des Benutzers weder lesen noch schreiben.
- Applets können nur mit dem Internet-Server kommunizieren, von dem die Webseite stammt, die das Applet enthält.
- Applets können keine Programme auf dem System des Benutzers ausführen.
- Applets können keine Programme auf der lokalen Plattform laden, einschließlich gemeinsam genutzter Bibliotheken wie DLLs.

Alle diese Regeln gelten für Java-Applets, die mit dem Netscape Navigator oder Microsoft Internet Explorer, die von den meisten Anwendern im Web heute bevorzugt werden, ausgeführt werden. Andere Java-Browser oder Tools ermöglichen eventuell eine Konfiguration des Sicherheitslevels. Auf diesem Weg könnten Sie Dateizugriffe auf bestimmte Ordner erlauben oder auch Netzwerkverbindungen zu ausgewählten Internet- Sites zulassen.

Der Applet-Viewer im JDK bietet z.B. Zugang zu einer Zugriffskontroll-Liste, in der festgelegt werden kann, welche Verzeichnisse ein Applet lesen oder beschreiben darf. Doch als Entwickler von Applets sollten Sie davon ausgehen, daß Ihr Publikum die Applets in einem Browser ablaufen läßt, der die strengsten Regeln für Applets vorsieht.

Für Java-Anwendungen gelten diese Beschränkungen nicht. Diese können die Möglichkeiten von Java voll ausschöpfen.



Obwohl es das Sicherheitsmodell von Java für bösartige Applets sehr schwer macht, auf dem System des Anwenders Schaden anzurichten, ist dies keine 100prozentige Sicherheit. Suchen Sie im Web nach dem Begriff »hostile applets«, und Sie werden eine Reihe von Beiträgen zum Thema Sicherheit in den verschiedenen Versionen von Java finden. Sie werden eventuell sogar Beispiel- Applets finden, die auf den Systemen der Anwender, die Java-fähige Browser verwenden, für Probleme sorgen. Java ist sicherer als andere Lösungen für die Web-Programmierung, wie z.B. ActiveX, dennoch sollten alle Anwender von Browsern sich mit diesem Thema vertraut machen.

Eine Java-Version wählen

Ein Java-Programmierer, der Applets schreibt, muß sich überlegen, welche Version von Java er dafür verwendet.

Zum Zeitpunkt des Erscheinens dieses Buches ist Java 1.02 die einzige Version der Sprache, die sowohl vom Netscape Navigator als auch vom Internet Explorer voll unterstützt wird. Beide Browser zusammen bringen es in der Welt der Applet-Anwender auf über 90% Marktanteil. Netscape war ziemlich langsam mit der vollen Integration von Java 1.1 in der Version 4.0 seines Browsers, und Microsoft wird diese Version eventuell nie unterstützen.



JavaSoft hat ein Add-on entwickelt, das sich Java Plug-In nennt und es Applet-Programmierern ermöglicht, die Erweiterungen von Java 1.1 und 1.2 in ihren Programmen zu verwenden. Im JDK 1.2 ist dieses Java Plug-In in der Version 1.2 enthalten. Die neuesten Informationen und Details dazu erfahren Sie unter der Web-Adresse:

<http://java.sun.com/products/plugin/index.html>

Wegen dieser Aufteilung scheint die folgende Vorgehensweise unter Programmierern sehr verbreitet zu sein:

- Applets werden so geschrieben, daß dabei nur Features von Java 1.02 zur Verwendung kommen, da diese so in allen Java-fähigen Browsern lauffähig sind.
- Applikationen werden mit Java 1.2 geschrieben, da sie auf jedem System mit einem Java-1.2-Interpreter ausgeführt werden können.

Java 1.2 wurde so entworfen, daß ein Programm, das nur Java-1.02- Features verwendet, von einem 1.02-Compiler erfolgreich kompiliert und von einem 1.02-Interpreter bzw. 1.02-fähigen Browser ausgeführt werden kann.

Sobald aber ein Applet irgendeines der Features von Java 1.1 oder Java 1.2 verwendet, kann das Programm nicht mehr von einem Browser ausgeführt werden, der diese Versionen der Sprache nicht unterstützt. Die einzige Testumgebung, die diese Versionen voll unterstützt, ist der neueste Applet-Viewer von JavaSoft.

Dies stellt eine häufige Fehlerquelle für Applet-Programmierer dar. Wenn Sie ein Java-1.2-Applet schreiben und es in einem Browser ausführen, der diese Version der Sprache nicht unterstützt, wie z.B. der Microsoft Internet Explorer 4.0, erhalten Sie Security- und Class-not-found-Fehler. Außerdem treten andere Probleme auf, die verhindern, daß das Applet ausgeführt wird.

In diesem Buch wird die Applet-Programmierung weitestgehend auf Basis der Techniken von Java 1.02 behandelt, da dies weiterhin den Standard für die Web-Programmierung darstellt. Applets sind normalerweise kleinere Programme, die viele der Erweiterungen der Sprache, die mit den Versionen 1.1 und 1.2 eingeführt wurden, überhaupt nicht brauchen.

Bei der Programmierung von Applikationen werden wir allerdings die neuesten und besten Features von Java 1.2 verwenden. Sie werden vielleicht auch mal in der Lage sein, die Klassenbibliothek von Java 1.2 in Ihren Applets zu verwenden, sobald für die Browser eine Möglichkeit gefunden wird, die Entwickler der Sprache einzuholen und mit diesen Schritt zu halten.

Die Unterschiede zwischen den Versionen sind in diesem Buch immer angemerkt. Der Java-Compiler wird Sie gelegentlich darauf hinweisen, wenn ein 1.02-Feature in Java 1.2 durch eine bessere Lösung ersetzt wurde. Entsprechend werden Sie auch gewarnt.

Erweiterte Kontrollmöglichkeiten über die Sicherheit

Das Sicherheitsmodell, das bis hierher beschrieben wurde, wurde mit der Version 1.02 eingeführt. Die aktuelle Version von Java bietet dem Web-Anwender eine Möglichkeit, einem Applet sein Vertrauen auszusprechen, so daß dieses ohne Einschränkungen auf dem System des Benutzers wie eine Applikation ausgeführt werden kann.

Java 1.2 bietet die Möglichkeit, die sehr spezifischen Sicherheitskontrollen für Applets und Applikationen festzulegen bzw. von diesen zu entfernen. Dieses Thema wird an Tag 17 behandelt.

Erstellen von Applets

Bei fast allen Java-Programmen, die Sie in den bisherigen Lektionen geschrieben haben, handelte es sich um Java-Anwendungen - einfache Programme mit einer `main()`-Methode-, die Objekte erstellt, Instanzvariablen setzt und Methoden ausführt.

Applets besitzen keine `main()`-Methode, die automatisch beim Start des Programms aufgerufen wird. Statt dessen gibt es einige Methoden, die an verschiedenen Punkten der Ausführung eines Applets aufgerufen werden. Über diese Methoden werden Sie heute einiges lernen.

Alle Applets sind Subklassen der Klasse `Applet` aus dem Paket `java.applet`. Die Klasse `Applet` stellt zwei Arten von Verhaltensweisen, die alle Applets haben müssen, zur Verfügung:

- Verhaltensweisen, damit das Applet als Teil des Browsers arbeitet und Ereignisse wie das Neuladen der Seite im Browser behandelt.
- Verhaltensweisen, um eine grafische Schnittstelle darstellen und Eingaben des Benutzers entgegennehmen zu können.

Obwohl ein Applet so viele andere Klassen wie nötig verwenden kann, ist die `Applet`- Klasse die Hauptklasse, die die Ausführung eines Applets steuert. Die Subklasse von `Applet`, die Sie erzeugen, hat die folgende Form:

```
public class yourApplet extends java.applet.Applet {
    // Code des Applet
}
```

Applets müssen generell als `public` deklariert werden, da die Klasse `Applet` selbst `public` ist. Diese Forderung gilt allerdings nur für die Hauptklasse des Applets. Alle Hilfsklassen können entweder `public` oder `private` sein. Mehr Informationen über diese Art der Zugriffskontrolle erhalten Sie an Tag 16.

Sobald der in einen Browser integrierte Java-Interpreter ein Java-Applet auf einer Webseite entdeckt, wird die Hauptklasse des Applet mitsamt aller verwendeten Hilfsklassen geladen. Der Browser erstellt automatisch eine Instanz der Klasse des Applets und ruft Methoden von `Applet` auf, sobald bestimmte Ereignisse eintreten.

Unterschiedliche Applets, die dieselbe Klasse verwenden, verwenden unterschiedliche Instanzen dieser Klasse. Aus diesem Grund können Sie mehrere Kopien desselben Applet-Typs auf einer Seite platzieren, die sich alle unterschiedlich verhalten können.

Wichtige Applet-Aktivitäten

Anstelle einer `main()`-Methode haben Applets Methoden, die beim Eintreten bestimmter Ereignisse während der Ausführung eines Applets aufgerufen werden.

Ein Beispiel für diese Methoden ist `paint()`, die immer dann aufgerufen wird, wenn der Inhalt des Applet-Fensters neu dargestellt werden muß.

Standard mäßig tun diese Methoden gar nichts. Die `paint()`-Methode z.B., die ein Applet von der Klasse `Applet` erbt, ist leer. Damit etwas im Applet-Fenster angezeigt wird, muß die `paint()`-Methode mit Verhaltensweisen überschrieben werden, die Text, Grafik oder andere Dinge anzeigen.

Sie lernen im weiteren Verlauf dieser Woche mehr über die verschiedenen Methoden, die in der Klasse `Applet` überschrieben werden können. Vorläufig erhalten Sie einen allgemeinen Überblick über fünf der wichtigeren Methoden der Ausführung eines Applets: *Initialisieren*, *Starten*, *Stoppen*, *Zerstören* und *Anzeigen*.

Initialisieren

Die Initialisierung eines Applets tritt auf, wenn ein Applet geladen wird. Sie kann die Erzeugung der Objekte, die

ein Applet benötigt, das Setzen eines Ausgangszustandes, das Laden von Bildern und Schriften oder das Setzen von Parametern umfassen. Um für die Initialisierung des Applets Verhaltensweisen zu definieren, überschreiben Sie die `init()`-Methode:

```
public void init() {
    //Code der Methode
}
```

Starten

Nach dem Initialisieren eines Applets wird es gestartet. Ein Applet kann auch gestartet werden, nachdem es zuvor gestoppt wurde. Ein Applet wird beispielsweise gestoppt, wenn der Benutzer einen Link zu einer anderen Seite verfolgt; kehrt er anschließend zur vorherigen Seite zurück, wird das Applet wieder gestartet.

Das Starten unterscheidet sich vom Initialisieren, denn während seines Lebenszyklus kann ein Applet mehrmals gestartet werden, während die Initialisierung nur einmal stattfindet. Um das Startverhalten für ein Applet festzulegen, überschreiben Sie die `start()`-Methode:

```
public void start() {
    // Code der Methode
}
```

In die `start()`-Methode lassen sich Funktionalitäten wie z.B. das Erstellen und Starten eines Thread, das Aussenden der entsprechenden Meldungen an Hilfsobjekte oder irgendeine andere Anweisung zur Ausführung des Startvorgangs unterbringen. Über das Starten von Applets erfahren Sie am 10. Tag noch mehr.

Stoppen

Stoppen und Starten gehen Hand in Hand. Ein Applet wird gestoppt, wenn der Benutzer die Seite mit dem Applet verläßt bzw. wenn das Applet manuell durch den Aufruf der `stop()`-Methode angehalten wird. Standardmäßig werden alle Threads, die das Applet gestartet hat, weiter ausgeführt (weitere Informationen zu Threads erhalten Sie am 10. Tag). Durch Überschreiben von `stop()` können Sie die Ausführung der Threads unterbrechen und dann neu starten, wenn das Applet wieder angezeigt wird. Die `stop()`-Methode hat die folgende Form:

```
public void stop() {
    // Code der Methode
}
```

Zerstören

Zerstören klingt etwas gewalttätig. Durch das Zerstören kann ein Applet hinter sich aufräumen, bevor es oder der Browser beendet wird, z.B. um eventuell laufende Threads zu beenden oder andere laufende Objekte freizugeben. Im allgemeinen brauchen Sie `destroy` nicht zu überschreiben, sofern Sie nicht bestimmte Ressourcen freigeben müssen, z.B. Threads, die das Applet erzeugt hat. Um einem Applet zu ermöglichen, hinter sich aufzuräumen, überschreiben Sie die `destroy()`-Methode.

```
public void destroy() {
    // Code der Methode
}
```



Sie werden sich eventuell fragen, wodurch sich `destroy()` von `finalize()` (siehe 7. Tag) unterscheidet? `destroy()` ist nur für Applets anwendbar. `finalize()` ist eine allgemeinere Art für ein einzelnes Objekt eines beliebigen Typs, hinter sich aufzuräumen.

Java verfügt über eine automatische Speicherfreigabe, den Garbage Collector, der für Sie die Speicherverwaltung übernimmt. Der Garbage Collector fordert Speicher von Ressourcen zurück, sobald ein Programm diese nicht

mehr verwendet. Aus diesem Grund müssen Sie normalerweise Methoden wie `destroy()` gar nicht verwenden.

Zeichnen

Das Zeichnen bestimmt, wie ein Applet Elemente auf dem Bildschirm ausgibt. Dabei kann es sich um Text, eine Linie, farbigen Hintergrund oder ein Bild handeln. Das Zeichnen kann Hunderte von Malen während des Lebenszyklus eines Applet vorkommen (z.B. einmal nach der Initialisierung des Applets, wenn das Browser-Fenster am Bildschirm vorübergehend deaktiviert und dann wieder in den Vordergrund geholt wird oder falls das Browser-Fenster an eine andere Position auf dem Bildschirm versetzt wird oder auch zyklisch im Falle einer Animation). Sie überschreiben die `paint()`-Methode, um einem Applet das Zeichnen am Bildschirm zu ermöglichen. Die `paint()`-Methode sieht wie folgt aus:

```
public void paint(Graphics g) {
    ... // Code der Methode
}
```

Beachten Sie, daß `paint()` im Gegensatz zu den anderen hier beschriebenen Methoden ein Argument besitzt: eine Instanz der Klasse `Graphics`. Dieses Objekt wird vom Browser erstellt und an `paint` übergeben, weshalb Sie sich darüber keine Gedanken machen müssen. Sie sollten jedoch sicherstellen, daß die `Graphics`-Klasse (Teil des Pakets `java.awt`) in den Applet-Code importiert wird. Dies wird zumeist mit der Anweisung `import` am Anfang der Java-Datei ausgeführt.

```
import java.awt.Graphics;
```



Wenn Sie viele Klassen aus demselben Paket, wie z.B. dem Abstract Windowing Toolkit, importieren, können Sie ein Platzhalterzeichen verwenden, um alle der Klassen in diesem Paket auf einmal zu laden. Die Anweisung `import java.awt.*;` macht z.B. alle public-Klassen des `java.awt`-Pakets verfügbar. Allerdings werden durch eine solche `import`-Anweisung keine Subklassen eines Pakets importiert. Das heißt, die Anweisung `import java.awt.*;` importiert z.B. nicht die Klassen aus dem Paket `java.awt.image`.

Ein einfaches Applet

Am 2. Tag haben Sie ein einfaches Applet namens `Palindrome` erstellt, das den Text "Go hang a salami, I'm a lasagna hog." ausgab. Sie haben das Applet als ein Beispiel für das Anlegen einer Subklasse erstellt und verwendet.

Wir gehen den Code für dieses Applet noch einmal durch, diesmal allerdings mit dem Blickpunkt auf jene Details, die Sie soeben über Applets gelernt haben. Listing 8.1 enthält den Code für dieses Applet:

Listing 8.1: Der komplette Quelltext von `Palindrome.java`

```
1: import java.awt.Graphics;
2: import java.awt.Font;
3: import java.awt.Color;
4:
5: public class Palindrome extends java.applet.Applet {
6:     Font f = new Font("TimesRoman", Font.BOLD, 36);
7:
8:     public void paint(Graphics screen) {
9:         screen.setFont(f);
10:        screen.setColor(Color.red);
11:        screen.drawString("Go hang a salami, I'm a lasagna hog.", 5, 40);
12:    }
13: }
```

Dieses Applet überschreibt `paint()`, eine der wichtigsten Methoden, die im vorherigen Abschnitt beschrieben wurden. Da das Applet eigentlich nichts besonders ausführt (nur ein paar Wörter am Bildschirm ausgibt) und es nichts zu initialisieren gibt, ist `start()`, `stop()` oder `init()` nicht erforderlich.

Die eigentliche Aufgabe dieses Applets (so gering sie auch sein mag) findet bei der `paint()`-Methode statt. Das an die `paint()`-Methode übergebene `Graphics`-Objekt enthält den Grafikstatus, d.h. die aktuellen Merkmale der Zeichenoberfläche. Dies beinhaltet z.B. Informationen über die aktuelle Schrift und Farbe, die für alle Zeichenoperationen verwendet werden.

Die Zeilen 9 und 10 definieren die Standardschrift und -farbe dieses Grafikstatus (das `Font`-Objekt in der Instanzvariablen `f` und ein Objekt, das die Farbe Rot darstellt und in der Klassenvariablen `red` der `Color`-Klasse gespeichert ist.)

Zeile 11 zeichnet den String "Go hang a salami, I'm a lasagna hog." in der angegebenen Schrift und Farbe auf Position 5, 40. Der Punkt 0 für `x`, `y` ist die obere linke Ecke der Zeichenfläche des Applets, wobei das positive `y` nach unten verläuft, so daß 40 der untere Rand des Applets ist. Abb. 8.1 zeigt, wie das Begrenzungsfeld und die Zeichenkette für das Applet auf der Seite gezeichnet werden.



Abbildung 8.1: Zeichnen eines Applets

Wenn Sie die korrekten Applet-Methoden in Ihrer Klasse (`init()`, `start()`, `stop()`, `paint()` usw.) implementieren, funktioniert das Applet auch ohne expliziten Ausgangspunkt problemlos.

Applet in eine Webseite einfügen

Nachdem Sie eine oder mehrere Klassen, die Ihr Applet enthalten, erstellt und wie bei jedem anderen Java-Programm in Klassendateien kompiliert haben, legen Sie mit HTML eine Webseite an, die dieses Applet aufnimmt.

Applets werden mit dem `<APPLET>`-Tag, einer HTML-Anweisung, die wie andere HTML-Elemente funktioniert, in eine Seite eingefügt. Es gibt auch eine große Menge von Entwicklungstools für Webseiten, wie z.B. Claris Homepage, Macromedia Dreamweaver oder Microsoft Frontpage, mit denen sich Applets auf einer Seite einfügen lassen, ohne direkt mit dem HTML-Code arbeiten zu müssen.

Das `<APPLET>`-Tag hat die Aufgabe, ein Applet in eine Webseite einzufügen und sein Erscheinungsbild in bezug auf andere Elemente der Seite zu kontrollieren.

Java-fähige Browser verwenden die Informationen, die in diesem Tag enthalten sind, um die kompilierten `.class`-Dateien des Applets zu finden und auszuführen. In diesem Abschnitt werden Sie lernen, wie Sie Java-Applets in eine Webseite einfügen und wie Sie die ausführbaren Dateien im Web zur Verfügung stellen.



Im folgenden Abschnitt wird davon ausgegangen, daß Sie zumindest Grundkenntnisse im Schreiben von

HTML-Seiten besitzen. Wenn Sie in diesem Bereich Hilfe benötigen, empfiehlt sich das Buch »HTML 4 in 14 Tagen« von Laura Lemay (erschienen bei Sams, ISBN 3-8272-2019-X).

Das Tag <APPLET>

Um ein Applet auf einer Webseite einzufügen, verwenden Sie das Tag <APPLET>, das von allen Browsern unterstützt wird, die Java-Programme ausführen können. Listing 8.2 zeigt ein einfaches Beispiel für eine Webseite mit einem Applet.

Listing 8.2: Der gesamte Quelltext von Palindrome.html

```

1: <HTML>
2: <HEAD>
3: <TITLE>The Palindrome Page</TITLE>
4: </HEAD>
5: <BODY>
6: My favorite meat-related palindrome is:
7: <BR>
8: <APPLET CODE="Palindrome.class" WIDTH=600 HEIGHT=100>
9: A secret if your browser does not support Java!
10: </APPLET>
11: </BODY>
12: </HTML>

```

In diesem Beispiel beinhaltet das Tag <Applet> drei Attribute:

- CODE, das den Namen der Hauptklasse des Applets angibt.
- WIDTH, das die Breite des Applet-Fensters auf der Webseite festlegt.
- HEIGHT, das die Höhe des Applet-Fensters angibt.

Die .class-Datei, die über das CODE-Attribut festgelegt wird, muß sich in demselben Ordner befinden, wie die Webseite, die das Applet beinhaltet, es sei denn Sie verwenden zusätzlich das Attribut CODEBASE, um einen anderen Ordner festzulegen. Sie werden später lernen, wie Sie dies tun.

Die Attribute WIDTH und HEIGHT müssen immer angegeben werden, da der Web- Browser wissen muß, wieviel Platz er dem Applet auf der Seite bereitstellen muß. Es kann leicht passieren, daß Sie in einem Programm auf einen Bereich außerhalb des Applet-Fensters zeichnen. Aus diesem Grund müssen Sie sicherstellen, daß Sie für das Applet-Fenster genügend Platz zur Verfügung stellen.

Text, Bilder und andere Elemente einer Webseite können zwischen die Tags <APPLET> und </APPLET> eingefügt werden. Diese Elemente werden nur angezeigt, wenn der Browser nicht in der Lage ist, Java-Programme zu verarbeiten. Dies ist eine gute Möglichkeit, den Benutzern mitzuteilen, daß sie ein Java-Applet verpassen, weil deren Browser diese Applets nicht unterstützt. Wenn Sie nichts zwischen den Tags <APPLET> und </APPLET> angeben, zeigen Browser, die Java nicht unterstützen, nichts anstelle des Applets an.

In dem aktuellen Beispiel wird über dem Applet der Text "My favorite meat-related palindrome is:" sichtbar. Anwender mit Java-fähigen Browsern sehen direkt unter diesem Text das Palindrome-Applet. Anwender mit Browsern, die Java nicht unterstützen, sehen den dafür vorgesehenen Alternativtext - "A secret if your browser does not support Java!".

Prüfen der Ergebnisse

Sobald Sie die Haupt-.class-Datei und eine HTML-Datei, die das Applet verwendet, haben, können Sie die HTML-Datei von Ihrer Festplatte in einen Java-fähigen Browser laden. Im Netscape Navigator verwenden Sie den Befehl Datei | Seite öffnen | Datei wählen, um lokale Dateien zu laden. Der entsprechende Befehl des Internet Explorer ist: Datei | Öffnen | Durchsuchen. Der Browser zeigt nun die HTML- Datei an, lädt dann Ihre Applet-Klasse und führt sie aus.

Wenn Sie nicht über einen Java-fähigen Browser verfügen, verwenden Sie zum Testen Ihres Applets die Tools, die häufig in der Entwicklungsumgebung angeboten werden. In JDK können Sie die Applets mit dem Applet-Viewer prüfen. Im Gegensatz zu einem Browser zeigt der Applet-Viewer nur die Applets, die sich auf einer Webseite befinden an. Die Webseite selbst zeigt er nicht an.

Abb. 8.2 zeigt die Seite Palindrome.html bei ihrer Ausführung in Netscape.

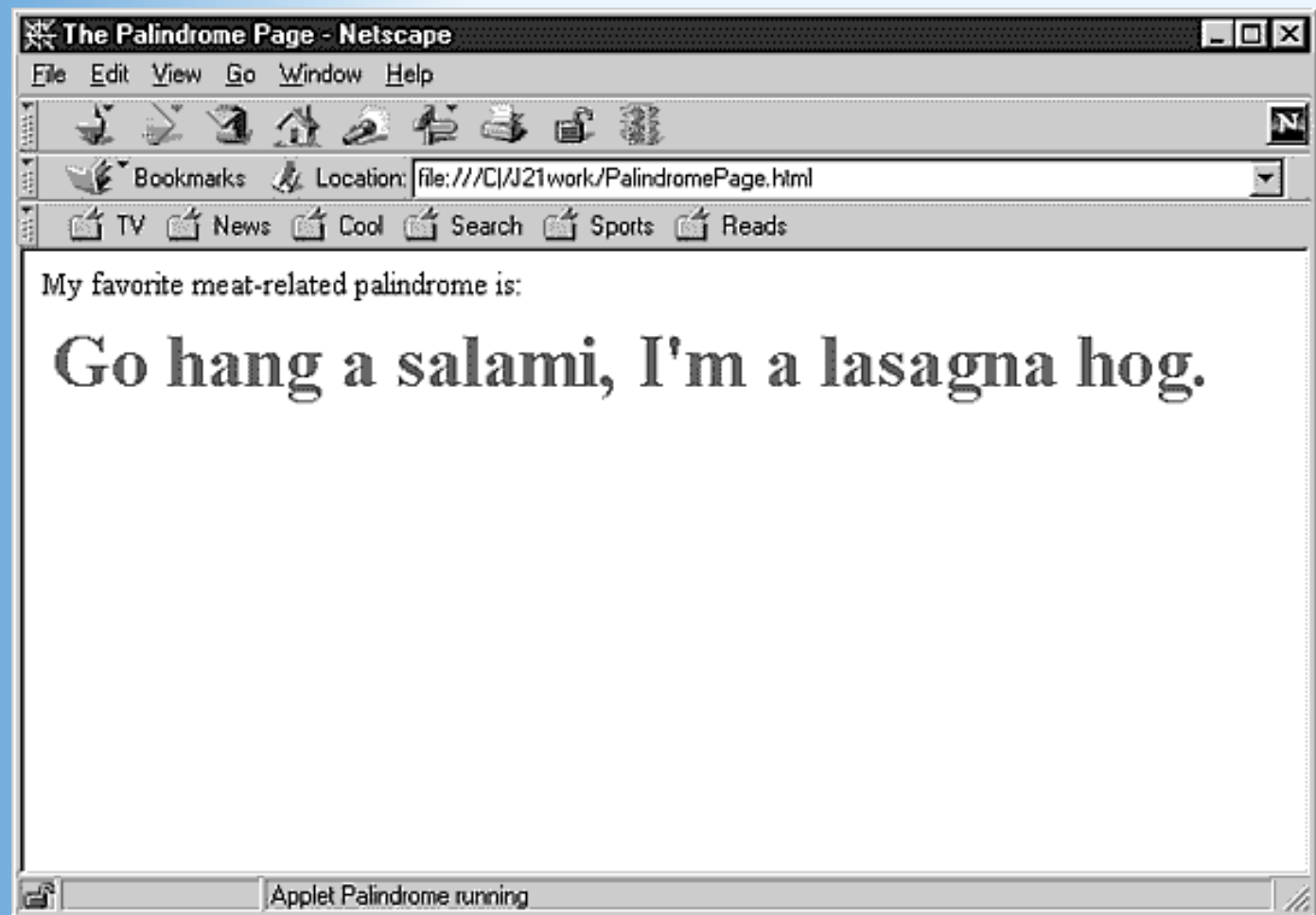


Abbildung 8.2:
Das Applet Hello Again

Java-Applets im Web bereitstellen

Nachdem Sie über ein Applet und eine HTML-Datei verfügen und sichergestellt ist, daß diese Elemente auf Ihrem lokalen System korrekt ausgeführt werden, können Sie das Applet dem Publikum im World Wide Web allgemein zur Verfügung stellen, so daß jeder mit einem Java-fähigen Browser das Applet anzeigen kann.

Java-Applets werden von einem Web-Server auf die gleiche Weise zur Verfügung gestellt wie HTML-Dateien, Bilder und andere Medien. Sie speichern das Applet in einem Ordner, der für den Web-Server zugänglich ist - oft derselbe Ordner, in dem sich auch die Webseite befindet, die das Applet beinhaltet. Der Web-Server sollte so konfiguriert sein, daß er Java-Applets Browsern bereitstellen kann, die die Sprache unterstützen.

Die folgenden Dateien sind die einzigen, die Sie auf den Server übertragen müssen:

- Die HTML-Seite, die das Applet beinhaltet.
- Alle .class-Dateien, die von dem Applet verwendet werden und nicht Teil der Standardklassenbibliothek von Java sind.

Wenn Sie wissen, wie Sie Webseiten, Bilddateien und andere Multimedia-Dateien publizieren, dann müssen Sie keine neuen Fähigkeiten erlernen, um Java-Applets auf Ihrer Website zu publizieren.

Weitere Informationen zum Tag <APPLET>

In seiner einfachsten Form benötigt das <APPLET>-Tag lediglich die Attribute CODE, WIDTH und HEIGHT, um ausreichend Platz für das Applet zu schaffen und anschließend das Applet in diesen Raum zu laden und dort auszuführen. Das <APPLET>-Tag unterstützt allerdings noch eine ganze Reihe anderer Attribute, die Ihnen dabei helfen, ein Applet in das gesamte Design einer Seite zu integrieren.



Die Attribute des <APPLET>-Tags entsprechen fast denen des -Tags.

Das Attribut ALIGN

Das Attribut ALIGN definiert, wie das Applet auf der Seite in Bezug auf andere Teile der Seite ausgerichtet wird. Dieses Attribut kann folgende neun Werte enthalten: LEFT, RIGHT, TOP, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM und ABSBOTTOM.

- ALIGN=LEFT richtet das Applet linksbündig an dem Text aus, der dem Applet auf der Seite folgt.
- ALIGN=RIGHT richtet das Applet rechtsbündig an dem Text aus, der dem Applet auf der Seite folgt.
- ALIGN=TEXTTOP richtet die Oberkante des Applets an dem höchsten Textelement in der Zeile aus.
- ALIGN=TOP richtet die Oberkante des Applets am höchsten Element in der Zeile aus (dies kann ein anderes Applet, ein Bild oder die Oberkante des Textes sein).
- ALIGN=ABSMIDDLE richtet die Mitte des Applets an der Mitte des größten Elements in der Zeile aus.
- ALIGN=MIDDLE richtet die Mitte des Applets an der Grundlinie des Textes aus.
- ALIGN=BASELINE richtet die Unterkante des Applets an der Grundlinie des Textes aus.
ALIGN=BASELINE entspricht ALIGN=ABSBOTTOM, ist aber als Bezeichnung wesentlich beschreibender.
- ALIGN=ABSBOTTOM richtet die Unterkante des Applets an dem niedrigsten Element in der Zeile aus (dies kann die Grundlinie des Textes, ein anderes Applet oder ein Bild sein).

Um die Formatierung der Ausrichtung, die mit dem ALIGN-Attribut festgelegt wurde, zu beenden, verwenden Sie das HTML-Tag
 mit dem CLEAR-Attribut. Dieses Attribut kennt drei Werte:

- <BR CLEAR=LEFT>: Die Anzeige des Rests der Webseite beginnt an der nächsten Stelle mit freiem linken Rand.
- <BR CLEAR=RIGHT>: Die Anzeige des Rests der Webseite beginnt an der nächsten Stelle mit freiem rechten Rand.
- <BR CLEAR=ALL>: Die Anzeige des Rests der Webseite beginnt an der nächsten Stelle mit freiem linken und rechten Rand.

Abb. 8.3 zeigt die verschiedenen Ausrichtungsoptionen. Der Smiley ist dabei ein Applet.



There is an applet to the left of this text.

There is an applet to the right of this text.



This applet lines up with the top edge of this sentence.



This applet lines up with the bottom edge of this sentence.



The middle of this applet lines up with the baseline of this sentence.



This applet lines up with the top text or image on this line.



This applet lines up with the middle of the largest text or image on this line.



This applet lines up with the bottom edge of the largest text or image on this line.



Abbildung 8.3: Ausrichtungsoptionen für Applets

Wenn Sie ein Webentwicklungstool verwenden, das es Ihnen ermöglicht, Java-Applets auf einer Seite zu platzieren, sollten Sie in der Lage sein, das ALIGN-Attribut mit LEFT, RIGHT oder einem der anderen Werte in diesem Programm zu setzen.

HSPACE und VSPACE

Die Attribute HSPACE und VSPACE werden dazu verwendet, den Abstand zwischen einem Applet und dem umliegenden Text in Pixeln anzugeben. HSPACE definiert den horizontalen Abstand links und rechts neben dem Applet. VSPACE bestimmt den vertikalen Abstand ober- und unterhalb des Applets. Als Beispiel dafür soll der folgende Auszug aus einem HTML-Code dienen, der einen vertikalen Abstand von 50 und einen horizontalen Abstand von 10 festlegt:

```
<APPLET CODE="ShowSmiley.class" WIDTH=45 HEIGHT=42
ALIGN=LEFT VSPACE=50 HSPACE=10>
Requires Java
</APPLET>
```

Die Abbildung 8.4 zeigt, wie dieses Applet, das einen Smiley auf weißem Hintergrund zeigt, mit anderen Elementen auf einer Webseite angezeigt würde. Der Hintergrund dieser Seite ist ein Raster, bei dem jede Zelle 10 mal 10 Pixel groß ist. Sie können dieses Raster dazu verwenden, den Abstand zwischen dem Applet und dem Text auf der Seite zu messen.



Abbildung 8.4:
Vertikaler und horizontaler Abstand

CODE und CODEBASE

Die letzten beiden Attribute für das Tag `<APPLET>` sind `CODE` und `CODEBASE`. Im Gegensatz zu den anderen Attributen des Tags wirken sich diese beiden nicht auf das Erscheinungsbild eines Applets aus, sondern geben an, wo sich die Haupt-.class-Datei und andere Dateien befinden. Sie werden von Java-fähigen Browsern verwendet, um diese Dateien auf einem Web Server zu finden und von diesem herunterzuladen.

`CODE` dient zur Bezeichnung des Namens der .class-Datei, in der sich das Applet befindet. Wird `CODE` allein im Tag `<APPLET>` verwendet, wird nach der Klassendatei in jenem Verzeichnis gesucht, in dem sich die HTML-Datei befindet, die auf das Applet verweist.

Beachten Sie, daß die Klassendateinamen, die in `CODE` angegeben werden, die Erweiterung .class haben müssen. Das folgende `<APPLET>`-Beispiel lädt ein Applet mit dem Namen `Bix.class` aus demselben Ordner, in dem sich auch die Webseite befindet:

```
<APPLET CODE="Bix.class" HEIGHT=40 WIDTH=400>
</APPLET>
```

Das Attribut `CODEBASE` wird verwendet, um den Browser anzuweisen, in einem anderen Ordner nach dem Applet und den Dateien, die es verwendet, zu suchen. `CODEBASE` legt einen anderen Ordner oder sogar eine andere Website fest, von wo die .class- Datei und die anderen Dateien geladen werden sollen. Der folgende Code lädt die .class-Datei `Bix.class` aus dem Ordner `Torshire`:

```
<APPLET CODE="Bix.class" CODEBASE="Torshire" HEIGHT=40 WIDTH=400>
</APPLET>
```

Im Anschluß sehen Sie ein Beispiel, bei dem die Java-.class-Datei von einer ganz anderen Website als der, auf der sich die Webseite befindet, geladen wird.

Das <OBJECT>-Tag

Das `<APPLET>`-Tag ist eine HTML-Erweiterung, die ausschließlich dafür eingeführt wurde, um Java-Programme in Webseiten einzufügen. Heute gibt es andere Arten von Programmen, die interaktiv ausgeführt werden können, darunter `ActiveX`-Steuerelemente, `NetRexx`-Applets und `Phyton`-Programme. Um mit all diesen Programmen arbeiten zu können, ohne daß für jedes ein eigenes Tag benötigt wird, wurde das `<OBJECT>` -Tag der HTML-Spezifikation hinzugefügt.

Das `<OBJECT>`-Tag wird für alle Objekte verwendet - interaktive Programme und andere externe Elemente -, die als Teil einer Webseite präsentiert werden können. Es wird von den Versionen 4.0 und höher des Netscape

Navigator und des Microsoft Internet Explorer unterstützt. Ältere Browser unterstützen dieses Tag nicht, so daß Sie in den meisten Fällen noch das <APPLET>-Tag verwenden werden.

Das <OBJECT>-Tag hat die folgende Form:

```
<OBJECT CLASSID="java:Bix.class" CODEBASE="javaclasses" HEIGHT=40 WIDTH=400>
</OBJECT>
```

Um statt des <APPLET>-Tags das <OBJECT>-Tag zu verwenden, sind die folgenden Änderungen nötig:

- Das <APPLET>-Tag ersetzen Sie durch das <OBJECT>-Tag.
- Das CODE-Attribut ersetzen Sie durch das Attribut CLASSID. Zusätzlich fügen Sie den Text "java:" vor den Namen der .class-Datei des Applets ein. Wenn das Applet z.B. GameApplet.class hieße, hätte das CLASSID-Attribut den Wert java:GameApplet.class

Die anderen Attribute bleiben gleich, einschließlich CODEBASE, HEIGHT, WIDTH und ALIGN. Im <OBJECT>-Tag können optional noch <PARAM>-Tags verwendet werden, die später noch beschrieben werden.

Das Listing 8.3 beinhaltet die Webseite, die das Palindrome-Applet über das <OBJECT>-Tag lädt. Alles andere entspricht dem vorigen Beispiel aus Listing 8.2.

Listing 8.3: Der gesamte Quelltext von Palindrome2.html

```
1: <HTML>
2: <HEAD>
3: <TITLE>The Palindrome Page</TITLE>
4: </HEAD>
5: <BODY>
6: <P>My favorite meat-related palindrome is:
7: <BR>
8: <OBJECT CLASSID="java:Palindrome.class" WIDTH=600 HEIGHT=100>
9: A secret if your browser does not support Java!
10: </OBJECT>
11: </BODY>
12: </HTML>
```

Java-Archive

Die Standardmethode zur Platzierung von Applets auf einer Webseite besteht darin, das Tag <APPLET> zur Definition der primären Klassendatei des Applets zu verwenden. Ein Java-fähiger Browser lädt das Applet dann herunter und führt es aus. Alle anderen vom Applet benötigten Klassen oder Dateien werden vom Web-Server heruntergeladen.

Das Problem bei der Ausführung von Applets mit dieser Methode ist, daß der Browser für jede einzelne Datei, die das Applet benötigt (das kann eine andere Hilfsklasse, ein Bild, eine Audiodatei, eine Textdatei etc. sein), eine eigene Verbindung zum Server herstellen muß. Da bereits für die Einrichtung der Verbindung selbst einige Zeit notwendig ist, wird der Zeitaufwand für das Laden des Applets und der zugehörigen Dateien durch diese Technik beträchtlich erhöht.

Die Lösung des Problems bietet ein Java-Archiv, d.h. eine JAR-Datei. Ein Java-Archiv ist eine Sammlung von Java-Klassen oder anderen Dateien, die in eine einzige Datei gepackt werden. Durch die Verwendung von Java-Archiven muß der Browser lediglich eine Verbindung zu diesem einen Archiv auf dem Server herstellen. Indem Sie die Anzahl der Dateien reduzieren, die vom Browser heruntergeladen werden müssen, läßt sich das Applet selbst schneller laden und ausführen. Java-Archive können auch komprimiert werden, wodurch sich die Dateigröße verringert und die Ladezeiten zusätzlich verkürzt werden (doch auch die Dekomprimierung durch den Browser kann einige Zeit beanspruchen).

Browser, die Java 1.1 oder höher unterstützen, unterstützen auch die JAR-Dateien; das JDK enthält ein Tool namens jar, mit dessen Hilfe sich Dateien in Java-Archive packen und wieder daraus entpacken lassen. JAR-Dateien können mit dem Zip-Format komprimiert oder ohne Komprimierung verpackt werden. Der folgende

Befehl packt alle Klassendateien und GIF-Bilddateien eines Verzeichnisses in ein einziges Java-Archiv mit dem Namen `Animate.jar`:

```
Jar cf Animate.jar *.class *.gif
```

Das Argument `cf` definiert zwei Befehlszeilen-Optionen, die sich für die Ausführung des Programms `jar` verwenden lassen. Die Option `c` gibt an, daß eine Java-Archivdatei erstellt werden soll, und die Option `f` legt fest, daß der Name der Archivdatei als nächstes Argument folgt.

Sie können auch nur bestimmte Dateien zu einem Java-Archiv hinzufügen, wie z.B. im folgenden:

```
jar cf Smiley.jar ShowSmiley.class ShowSmiley.html spinhead.gif
```

Dieses Kommando erzeugt ein Archiv mit dem Namen `Smiley.jar`, das drei Dateien beinhaltet: `ShowSmiley.class`, `ShowSmiley.html` und `spinhead.gif`.

Wenn Sie `jar` ohne Argumente ausführen, sehen Sie eine Liste der verfügbaren Optionen.

Nachdem ein Java-Archiv erstellt ist, wird das Attribut `ARCHIVES` mit dem Tag `<APPLET>` verwendet, um anzugeben, wo sich das Archiv befindet. Sie können Java- Archive in einem `<APPLET>`-Tag wie folgt benutzen:

```
<applet code="ShowSmiley.class" archives="Smiley.jar" width=45 height=42>
</applet>
```

Dieses Tag gibt an, daß das Archiv `Smiley.jar` Dateien enthält, die vom Applet benötigt werden. Browser und Surf-Tools, die JAR-Dateien unterstützen, suchen dann innerhalb des Archivs nach den für das Applet notwendigen Dateien.



Ein Java-Archiv kann zwar Klassendateien enthalten, aber auch bei der Verwendung des Attributs `ARCHIVES` muß das Attribut `CODE` verwendet werden. Der Browser muß auch in diesem Fall den Namen der Hauptklassendatei des Applets kennen, um diese laden zu können.

Weitere Archiv-Formate

Ehe die Entwickler von Java das Dateiformat JAR einführten, haben sowohl Netscape als auch Microsoft eigene Archivlösungen angeboten. Diese bieten gegenüber den Java-Archiven keine Vorteile, aber sie lassen sich ebenfalls mit Browsern verwenden, die die Java Versionen über 1.02 nicht unterstützen.

Die aktuellen Versionen des Netscape-Browsers unterstützten ZIP-Archive mit dem Attribut `ARCHIVE`, diese lassen sich jedoch nur für Klassendateien einrichten, nicht für Bilder oder andere vom Applet benötigte Dateien. In Netscape verwenden Sie das Attribut `ARCHIVE` dazu, den Namen des Archivs anzugeben:

```
<APPLET CODE="MyApplet.class" ARCHIVE="appletstuff.zip" WIDTH=100 HEIGHT=100>
</APPLET>
```

Das Archiv selbst ist eine unkomprimierte Zip-Datei. Die üblichen Zip-Dateien, die durch Komprimierung Dateigrößen reduzieren, werden nicht erkannt. Hilfsklassen können sich inner- oder außerhalb der Zip-Datei befinden; der Browser von Netscape sucht an beiden Positionen. Das Attribut `ARCHIVE` wird von anderen Browsern oder Applet-Viewern ignoriert.

Der Microsoft Internet Explorer erkennt einen dritten Typ von Archivformaten für Java-Applets: die CAB-Datei. *CAB* ist die Abkürzung für *Cabinet* und bietet eine Möglichkeit, Dateien zusammenzufassen und für eine schnellere Übertragung im Web zu komprimieren.

Cabinet-Archive werden mit einem Tool von Microsoft erstellt, welches `CABarc` heißt. Es steht aktuell zum Herunterladen unter folgender Adresse zur Verfügung:

<http://www.microsoft.com/workshop/prog/cab/>

Mit CABarc können Sie alle Klassendateien und alle anderen für das Applet erforderlichen Dateien in einem einzigen Archiv komprimieren. Dieses Archiv hat die Dateierweiterung .CAB. Um das Archiv anzugeben, wird der Parameter cabbase zusammen mit dem Tag <PARAM> in HTML verwendet. Der Wert von cabbase ist gleich dem Namen der .CAB-Datei. Hier ein Beispiel:

```
<APPLET CODE="DanceFever.class" WIDTH=200 HEIGHT=450>
<PARAM NAME="cabbase" VALUE="DanceFever.cab">
</APPLET>
```

Ebenso wie das Attribut ARCHIVE wird der Parameter cabbase von Web-Browsern ignoriert, die diesen nicht unterstützen.

Die Archivfunktionen von Netscape und Microsoft wurden vor der Herausgabe der Version Java 1.1 eingeführt, deshalb funktionieren sie mit den aktuellen Versionen dieser Browser und eventuell anderen Browsern. Wenn Sie eine dieser Lösungen bevorzugen, sollten Sie beide Archivformen und einen Satz der einzelnen Dateien auf Ihrem Web-Server abspeichern. Auf diese Weise können alle Java-fähigen Browser das Applet verwenden.

Parameter an Applets weitergeben

Bei Java-Anwendungen können Sie Parameter an die main()-Methode weitergeben, indem Sie in der Befehlszeile Argumente verwenden. Sie können diese Argumente dann innerhalb des Körpers der Klasse analysieren lassen, damit sich die Anwendung den definierten Argumenten entsprechend verhält.

Applets verfügen jedoch nicht über eine Befehlszeile. Wie lassen sich verschiedene Argumente an ein Applet weiterleiten? Applets können von der HTML-Datei, die das Tag <APPLET> enthält, durch Verwendung von Applet-Parametern verschiedene Eingaben erhalten. Um die Parameter in einem Applet einzurichten und zu handhaben, benötigen Sie folgende Elemente:

- Ein spezielles Parameter-Tag in der HTML-Datei
- Code im Applet, der diese Parameter analysiert

Applet-Parameter bestehen aus folgenden zwei Teilen: dem Parameternamen, der eine von Ihnen gewählte Bezeichnung ist, und dem Wert, welcher dem tatsächlichen Wert dieses speziellen Parameters entspricht. Sie können also beispielsweise die Farbe des Textes in einem Applet definieren, indem Sie einen Parameter mit dem Farbnamen und dem Wert red (rot) angeben. Oder Sie legen die Geschwindigkeit einer Animation fest, indem Sie einen Parameter mit dem Namen Speed und einen Wert von 5 verwenden.

In der HTML-Datei, die das eingebettete Applet enthält, fügen Sie Parameter mit dem Tag <PARAM> ein. Dieses Tag verfügt über zwei Attribute für den Namen und den Wert, diese haben die Bezeichnung Name und Value. Das Tag <PARAM> befindet sich innerhalb der Tags <APPLET> und </APPLET>:

```
<APPLET CODE="QueenMab.class" WIDTH=100 HEIGHT=100>
<PARAM NAME=font VALUE="TimesRoman">
<PARAM NAME=size VALUE="24">
A Java applet appears here.
</APPLET>
```

In diesem Beispiel werden zwei Parameter für das Applets QueenMab bestimmt: font mit dem Wert TimesRoman und size mit dem Wert 24.

Diese Parameter werden beim Laden des Applets weitergeleitet. In der init()-Methode des Applets können Sie diese Parameter mit der getParameter()-Methode berücksichtigen. getParameter() nimmt ein Argument an - eine Zeichenkette, die den Namen des Parameters bezeichnet, nach dem Sie suchen. Dann gibt die Methode eine Zeichenkette mit dem entsprechenden Wert aus. (Wie bei Argumenten in Java- Anwendungen werden alle Parameterwerte in Zeichenketten konvertiert.) Um den Wert des font-Parameters aus der HTML-Datei zu erhalten, können Sie z.B. die folgende Zeile in die init()-Methode aufnehmen:

```
String theFontName = getParameter("font");
```




Die Namen der in `<PARAM>` angegebenen Parameter und die Namen der Parameter in `getParameter` müssen absolut identisch sein, auch in bezug auf Groß- und Kleinschreibung. Mit anderen Worten: `<PARAM=NAME="eecum-mings">` unterscheidet sich von `<PARAM=NAME="EECumming">`. Werden Ihre Parameter nicht richtig an das Applet weitergeleitet, überprüfen Sie die Parameternamen.

Falls ein erwarteter Parameter nicht in der HTML-Datei angegeben wurde, gibt `getParameter()` Null zurück. In den meisten Fällen wird auf einen null-Parameter getestet und ein vernünftiger Standardwert zurückgegeben.

```
if (theFontName == null)
theFontName = "Courier"
```

Bedenken Sie außerdem, daß diese Methode Strings zurückgibt. Wenn Sie einen Parameter eines anderen Typs benötigen, müssen Sie den übergebenen Parameter selbst konvertieren. Nehmen Sie z.B. die HTML-Datei für das QueenMab-Applet. Um den size-Parameter zu parsen und ihn der Integer-Variablen `theSize` zuzuweisen, werden Sie eventuell den folgenden Quellcode verwenden:

```
int theSize;
String s = getParameter("size");
if (s == null)
theSize = 12;
else theSize = Integer.parseInt(s);
```

Noch nicht ganz klar? Dann wollen wir ein Beispiel für ein Applet schreiben, das diese Technik verwendet. Sie modifizieren das Palindrome-Applet dahingehend, daß es ein bestimmtes Palindrom ausgibt, z.B. »Dennis and Edna sinned" oder "No, sir, prefer prison". Der Name wird über einen HTML-Parameter an das Applet weitergeleitet. Das Projekt wird den Namen `NewPalindrome` tragen.

Kopieren Sie dazu die ursprüngliche Palindrome-Klasse, und benennen Sie diese entsprechend um, wie in Listing 8.4 dargestellt:

Listing 8.4: Der Anfang von `NewPalindrome.java`

```
1: import java.awt.Graphics;
2: import java.awt.Font;
3: import java.awt.Color;
4:
5: public class NewPalindrome extends java.applet.Applet {
6:     Font f = new Font("TimesRoman", Font.BOLD, 36);
7:
8:     public void paint(Graphics screen) {
9:         screen.setFont(f);
10:        screen.setColor(Color.red);
11:        screen.drawString("Go hang a salami, I'm a lasagna hog.", 5, 40);
12:    }
13: }
```

Als erstes müssen Sie in dieser Klasse Platz für den Palindrome-Parameter schaffen, um diesen zu speichern. Da dieser Name im gesamten Applet benötigt wird, verwenden wir eine Instanzvariable unmittelbar nach der Variablen für den Font:

```
String palindrome;
```

Um dieser Variablen einen Wert zuzuweisen, müssen Sie den Parameter aus der HTML-Datei holen. Die beste Stelle, um Applet-Parameter zu verarbeiten, ist die `init()`-Methode. Die `init()`-Methode wird ähnlich definiert wie `paint()` (public ohne Argumente und void als Rückgabetyt). Achten Sie darauf, daß Sie Parameter auch auf einen null-Wert testen. Wird kein Palindrom angegeben, ist der Standard in diesem Fall »Dennis and Edna sinned", wie Sie im folgenden sehen können:

```
public void init() {
    palindrome = getParameter("palindrome");
    if (palindrome == null)
        palindrome = "Dennis and Edna sinned";
}
```

Nun muß nur noch die paint()-Methode geändert werden, damit der neue Parametername verwendet wird. Die ursprüngliche sieht wie folgt aus:

```
screen.drawString("Go hang a salami, I'm a lasagna hog.", 5, 50);
```

Um den neuen String, den Sie in der Instanzvariablen palindrome gespeichert haben, zu zeichnen, müssen Sie nur das String-Literal durch die Variable ersetzen:

```
screen.drawString(palindrome, 5, 50);
```

Listing 8.5 zeigt das endgültige Ergebnis der Klasse NewPalindrome. Kompilieren Sie es, um eine komplette Klassendatei zu erhalten.

Listing 8.5: Der gesamte Quelltext von NewPalindrome.java

```
1: import java.awt.Graphics;
2: import java.awt.Font;
3: import java.awt.Color;
4:
5: public class NewPalindrome extends java.applet.Applet {
6:     Font f = new Font("TimesRoman", Font.BOLD, 36);
7:     String palindrome;
8:
9:     public void paint(Graphics screen) {
10:         screen.setFont(f);
11:         screen.setColor(Color.red);
12:         screen.drawString(palindrome, 5, 50);
13:     }
14:
15:     public void init() {
16:         palindrome = getParameter("palindrome");
17:         if (palindrome == null)
18:             palindrome = "Dennis and Edna sinned";
19:     }
20: }
```

Nun erstellen wir eine HTML-Datei, die dieses Applet enthält. Listing 8.6 zeigt eine neue Webseite für das NewPalindrome-Applet.

Listing 8.6: Der gesamte Quelltext von NewPalindrome.html

```
1: <HTML>
2: <HEAD>
3: <TITLE>The New Palindrome Page</TITLE>
4: </HEAD>
5: <BODY>
6: <P>
7: <APPLET CODE="NewPalindrome.class" WIDTH=600 HEIGHT=100>
8: <PARAM NAME=palindrome VALUE="No, sir, prefer prison">
9: Your browser does not support Java!
10: </APPLET>
11: </BODY>
12: </HTML>
```

Achten Sie darauf, daß das Tag <APPLET>, welches auf die Klassendatei für das Applet verweist, die passende Breite und Höhe (200 und 50) angibt. Unmittelbar darunter (in Zeile 8) steht das Tag <PARAM>, welches zur Weitergabe an den Namen verwendet wird. Hier ist der NAME-Parameter einfach palindrome, und der Wert ist "No, sir, prefer prison".

Wenn Sie diese HTML-Datei in den Netscape Navigator laden, erscheint das Ergebnis aus Abb. 8.5.

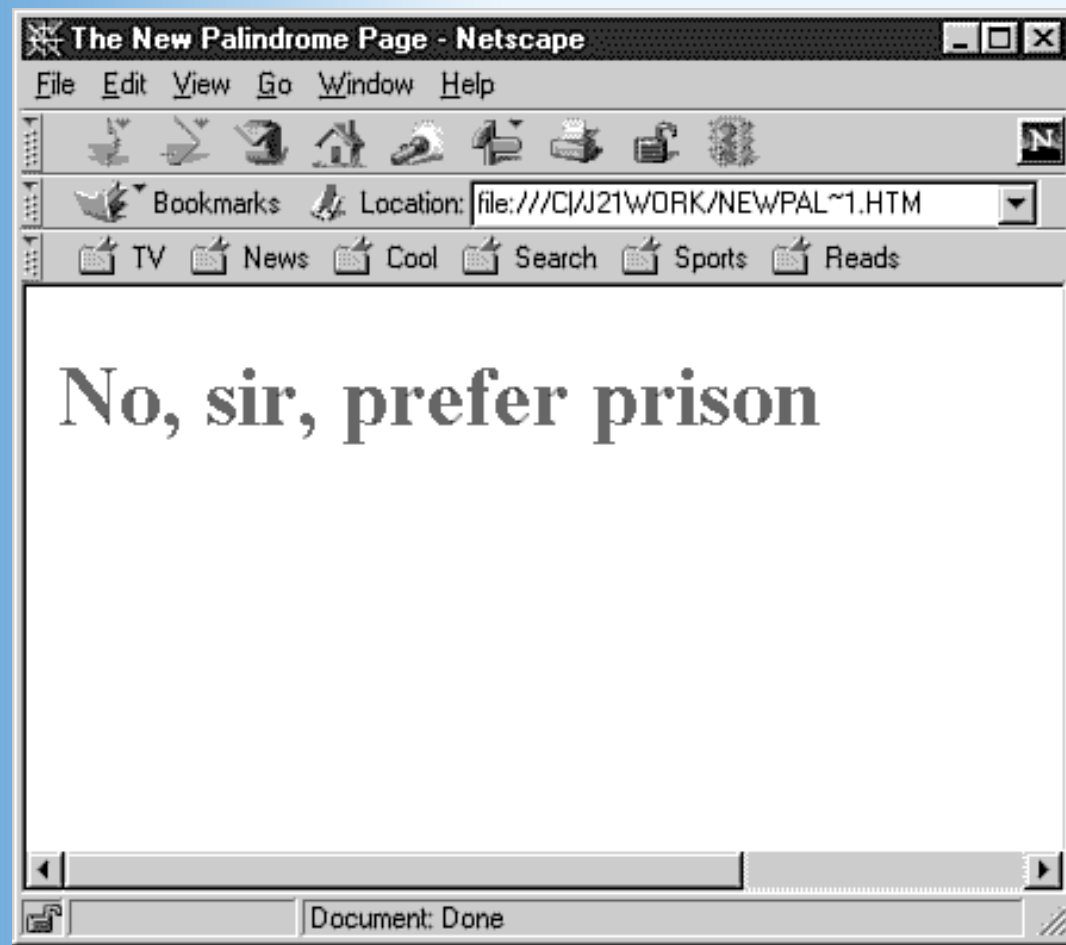


Abbildung 8.5:
Die Seite NewPalindrome.html im Netscape Navigator

Im Code von NewPalindrome ist der String Dennis and Edna sinned als Standardwert für den Fall festgelegt, daß kein Palindrom angegeben ist. Das Listing 8.7 stellt eine HTML-Datei dar, die keine Parameter übergibt.

Listing 8.7: Der gesamte Quelltext von NewPalindrome2.html

```

1: <HTML>
2: <HEAD>
3: <TITLE>Hello!</TITLE>
4: </HEAD>
5: <BODY>
6: <P>
7: <APPLET CODE="New Palindtrome.class" WIDTH=600 HEIGHT=10>
8: Your browser does not support Java!
9: </APPLET>
10: </BODY>
11: </HTML>

```

Da hier kein Palindrom angegeben ist, verwendet das Applet den Standard, und das Ergebnis erscheint wie in Abb. 8.6 dargestellt.

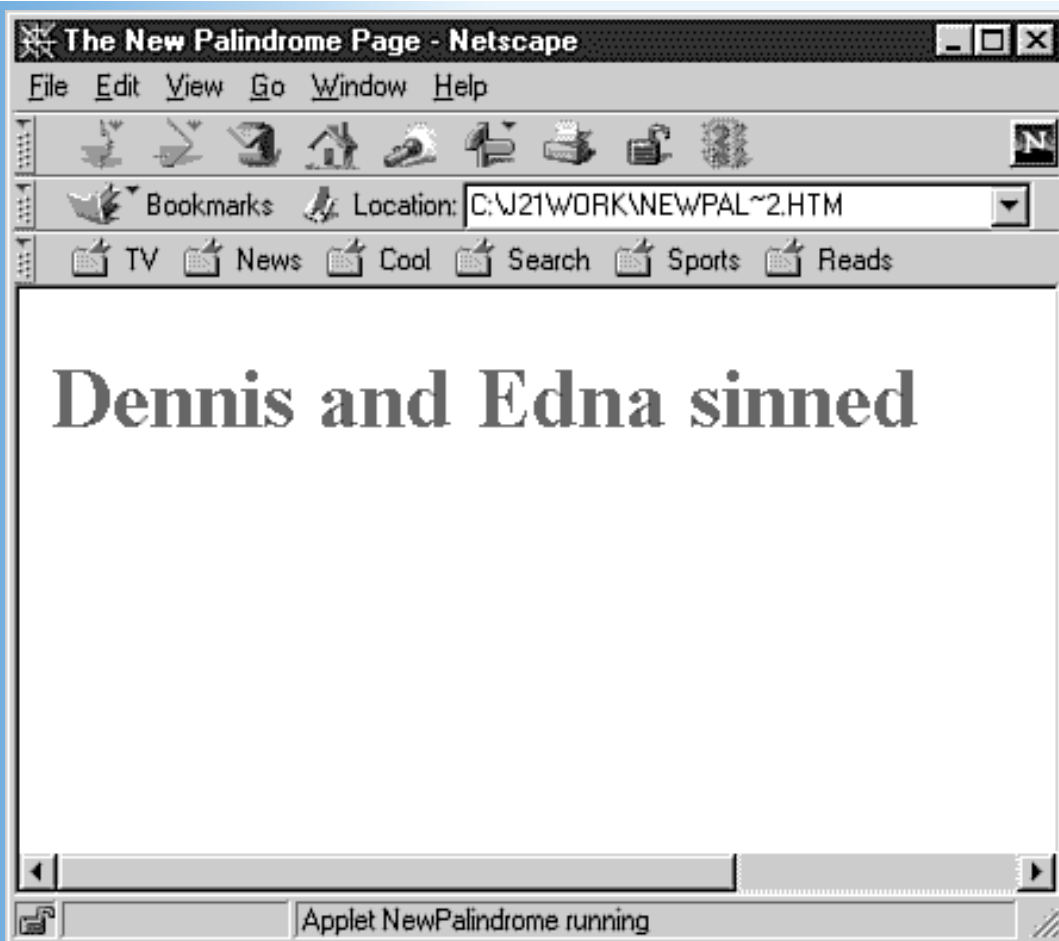


Abbildung 8.6:
NewPalindrome2.html im Netscape Navigator

Zusammenfassung

Man kann darüber streiten, ob Applets auch heute noch den Hauptbereich der Java- Entwicklung darstellen - mehr als zwei Jahre, nachdem die Sprache zum erstenmal veröffentlicht wurde.

Allerdings bleiben Applets das Anwendungsgebiet für Java, mit dem am meisten Menschen in Berührung kommen, da auf Tausenden von Websites Applets verwendet werden. Schenkt man Altavista (<http://www.altavista.com>) Glauben, dann gibt es mehr als 900.000 Webseiten, die Applets beinhalten.

Da sie in Webseiten ausgeführt und angezeigt werden, können Applets die Grafik, die Benutzerschnittstelle und die Ereignisstruktur des Web-Browsers verwenden. Diese Möglichkeiten bieten dem Applet-Programmierer eine große Menge an Funktionalität ohne große Schufterei.

Heute haben Sie die Grundlagen der Applet-Erstellung erlernt. Darunter auch die folgenden Themen:

- Alle Applets, die Sie mit Java schreiben, sind Subklassen der Klasse `java.applet.Applet`. Die Applet-Klasse bietet grundlegende Eigenschaften und Verhaltensweisen dafür, daß das Applet in einem Browser ausgeführt werden kann.
- Applets haben fünf Hauptmethoden, die für grundlegende Aktivitäten eines Applets während seines Lebenszyklus verwendet werden: `init()`, `start()`, `stop()`, `destroy()` und `paint()`. Diese Methoden werden überschrieben, um bestimmte Funktionalitäten in einem Applet zu bieten.
- Applets binden Sie über das Tag `<APPLET>` in eine HTML-Webseite ein. Alternativ können Sie dafür auch ein Webentwicklung-Tool verwenden. Trifft ein Java-fähiger Browser auf `<APPLET>`, lädt er das Applet und zeigt es entsprechend der Tag- Beschreibung an.
- Um das Herunterladen von Applets auf eine Webseite zu beschleunigen, können Sie drei Lösungen ins Auge fassen: Java-Archivdateien, die mit Java-1.1-Browsern funktionieren und alle Dateien in einem einzigen Archiv komprimieren können, die ein Applet erfordert. Ferner stehen die Zip-Archive für Netscape zur

Verfügung, die jedoch nur für Klassendateien eingesetzt werden können und keine Komprimierung vorsehen. Drittens können Sie die Cabinet-Dateien von Microsoft verwenden, die ebenso wie die Java-Archivdateien vom aktuellen Internet Explorer benutzt werden.

- Applets können von einer Webseite Informationen über das <PARAM>-Tag erhalten. Im Rumpf des Applets können Sie auf diese Parameter mit der Methode `getParameter()` zugreifen.

Fragen und Antworten

Frage:

Ich habe ein Applet, das Parameter holt, und eine HTML-Datei, die diese Parameter weitergibt. Beim Ausführen meines Applets erhalte ich aber nur Nullwerte. Woran liegt das?

Antwort:

Stimmen die Namen der Parameter (im NAME-Attribut) genau mit denen überein, die Sie zum Prüfen von `getParameter()` verwenden? Sie müssen absolut übereinstimmen, auch hinsichtlich der Groß- und Kleinschreibung. Achten Sie darauf, daß Ihre <PARAM>-Tags innerhalb von öffnenden und schließenden <APPLET>-Tags stehen und daß kein Name falsch geschrieben wurde.

Frage:

Wie kann ich, da Applets nicht über eine Befehlszeile verfügen, eine einfache Debugging-Ausgabe wie `System.out.println()` in einem Applet vornehmen?

Antwort:

Dies ist trotzdem möglich. Je nach Browser oder Java-fähiger Umgebung verfügen Sie über ein Konsolenfenster, in dem die Debugging-Ausgabe (wie `System.out.println()`) erscheint, oder sie wird in einer Log-Datei gespeichert (Netscape verfügt im Menü Optionen über eine Java-Konsole, der Internet Explorer verwendet eine Java-Log-Datei, die Sie durch Optionen/Erweitert aktivieren können). Sie können in den Applets weiterhin Meldungen mit `System.out.println()` ausgeben, sollten diese aber anschließend entfernen, damit Sie den Anwender nicht verwirren!

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).

Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 2**Tag 9**

Programme mit Grafik, Fonts und Farbe verfeinern

Die beste Möglichkeit, einen nichtprogrammierenden Bekannten zu beeindrucken, ist ein Programm, das Grafik verwendet. Onkel Walter wird wahrscheinlich die Nuancen einer wohlgeformten for-Schleife oder einer eleganten Klassenhierarchie nicht zu schätzen wissen; zeigen Sie ihm eine Animation mit einem Kleinkind, das den Ententanz aufführt, und er wird von Ihren Programmierfähigkeiten beeindruckt sein.

Heute beginnen Sie damit zu lernen, wie Sie Freunde gewinnen und Leute beeinflussen, indem Sie Applets schreiben, die Grafik, Fonts und Farbe verwenden.

Um grafische Features in einem Programm zu verwenden, benutzen Sie die Klassen des java.awt-Paketes, das die meisten der visuellen Möglichkeiten von Java liefert. Mit diesen Klassen geben Sie in einem Applet Text aus und zeichnen Figuren, wie z.B. Kreise und Polygone. Sie werden auch lernen, unterschiedliche Schriften und Farben für die Figuren, die Sie zeichnen, zu verwenden.

Sie werden auch anfangen, mit den verbesserten Zeichenmöglichkeiten von Java2D, einem Satz von Klassen, der mit Java 1.2 eingeführt wurde, zu arbeiten. Diese Klassen bieten einige bemerkenswerte Features:

- Objekte mit Anti-Aliasing
- Objekte mit Verlaufsfüllung
- Linien mit unterschiedlicher Stärke zeichnen

Die Klasse Graphics

Sie können sich ein Applet als eine Leinwand für grafische Operationen vorstellen. Sie haben bereits die Methode drawString() verwendet, um Text in einem Applet auszugeben - man könnte auch sagen: den Text im Ausgabebereich des Applets zu zeichnen. Die Farbe und die Schriftart und die Farbe der Schrift wurden ausgewählt, bevor die einzelnen Zeichen ausgegeben wurden. Dies ist in etwa so wie bei einem Künstler, der die Farbe und den Pinsel wählt, bevor er mit dem Malen beginnt.

Text ist allerdings nicht das einzige, was Sie in einem Applet-Fenster zeichnen können. Sie können Linien, Ovale, Kreise, Bögen, Rechtecke und andere Polygone zeichnen.

Die meisten der elementaren Zeichenoperationen sind Methoden, die in der Klasse Graphics definiert sind. In einem Applet müssen Sie kein Objekt der Klasse Graphics erzeugen, um etwas zeichnen zu können - wie Sie sich vielleicht erinnern werden, wird der Methode paint() ein Objekt der Klasse Graphics als Parameter übergeben. Dieses Objekt repräsentiert das Applet-Fenster. Mit den Methoden dieses Objekts zeichnen Sie in das Applet-Fenster.

Die Graphics-Klasse ist Bestandteil des Paketes java.awt. Deshalb müssen alle Applets, die etwas zeichnen, über das import-Statement die Klasse Graphics in dem Programm verfügbar machen.

Listing 9.1 ist ein einfaches Applet, das mit der drawString()-Methode Text ausgibt, wie Sie das bereits zuvor in dem Palindrom-Applet gemacht haben.

Listing 9.1: Der Anfang von Map.java

```

1: import java.awt.Graphics;
2:
3: public class Map extends java.applet.Applet {
4: public void paint(Graphics screen) {
5:     screen.drawString("Florida", 185, 75);
6: }
7: }

```

Dieses Applet verwendet die Methode `drawString()` des `screen`-Objekts, um den String »Florida« bei den Koordinaten 185,75 auszugeben. Listing 9.2 zeigt den HTML-Quelltext, über den das Applet angezeigt wird, nachdem es zu einer `.class`-Datei kompiliert wurde.

Listing 9.2: Der Quelltext von Map.html

```

1: <body bgcolor="#c4c4c4">
2: <div align="center">
3: <applet code="Map.class" height=350 width=350>
4: </applet>
5: </div>
6: </body>

```

In Abbildung 9.1 sehen Sie die Seite mit dem Applet im Netscape Navigator.

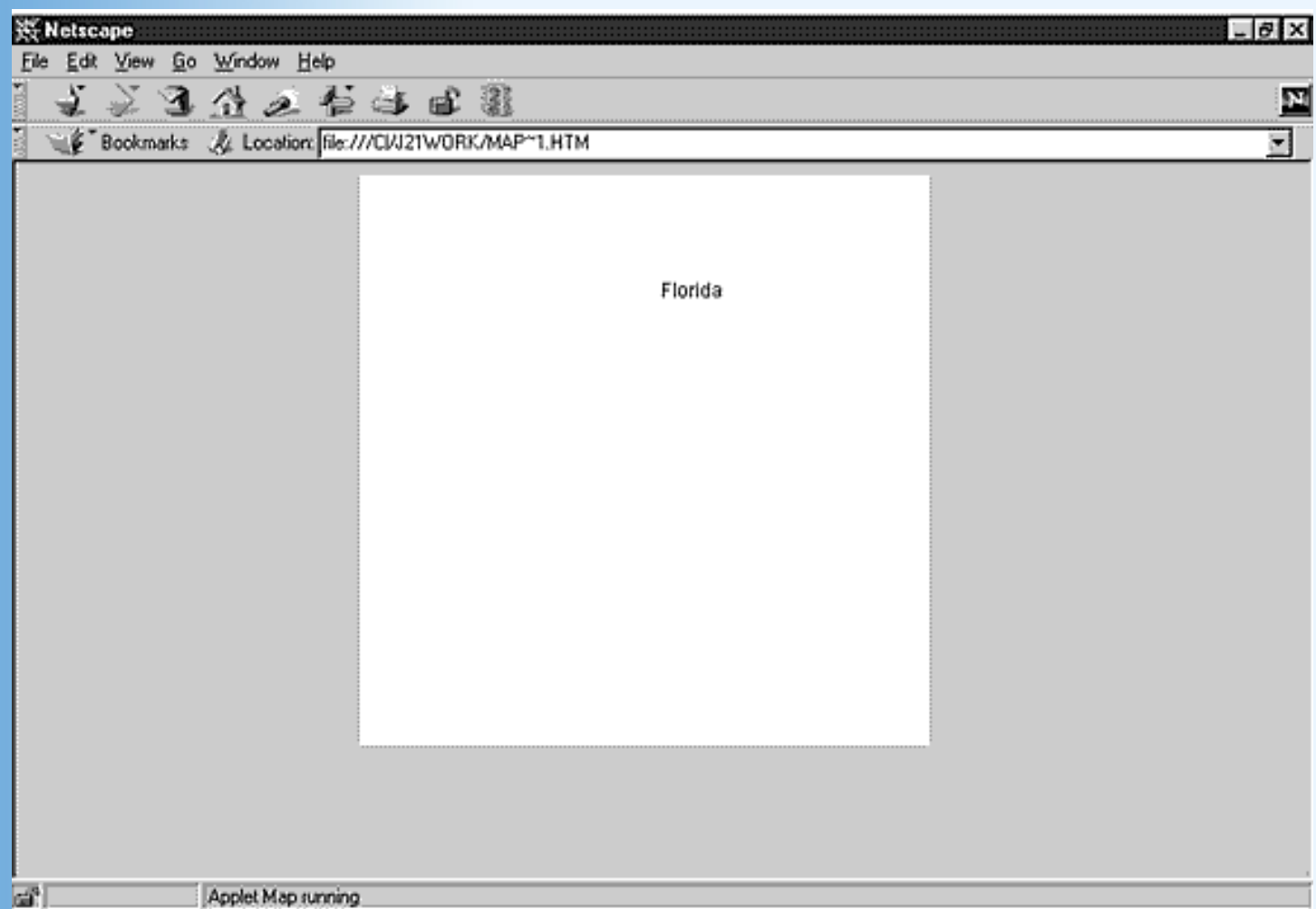


Abbildung 9.1:
Text im Applet-Fenster ausgeben

Alle der elementaren Zeichenbefehle, die Sie heute kennenlernen werden, sind Methoden der Klasse `Graphics`, die in der `paint()`-Methode eines Applets aufgerufen werden. Dies ist der ideale Ort für alle Zeichenoperationen, da `paint()` automatisch aufgerufen wird, sobald das Applet-Fenster neu dargestellt werden

muß. Wenn z.B. das Fenster eines anderen Programms das Applet-Fenster verdeckt, muß das Applet-Fenster neu dargestellt werden, sobald das Fenster des anderen Programms verschoben wird. Indem Sie alle Zeichenoperationen in die `paint()`-Methode einfügen, stellen Sie sicher, daß kein Teil der Ausgabe beim Neuzeichnen übergangen wird.

Sie werden mit jeder neu behandelten Zeichenmethode das Map-Applet erweitern.

Das Koordinatensystem von Graphics

Wie bereits die `drawString()`-Methode haben alle Zeichenmethoden Argumente, die die x,y-Koordinaten für die jeweilige Aktion angeben. Manche erwarten mehr als ein Koordinaten-Paar. Dies ist z.B. bei Linien der Fall, bei denen ein x,y-Koordinatenpaar den Anfangspunkt angibt und ein anderes x,y-Koordinatenpaar den Endpunkt.

Javas Koordinatensystem verwendet Pixel als Maßeinheit. Der Koordinatenursprung (0,0) liegt in der oberen linken Ecke des Applet-Fensters. Die x-Werte wachsen nach rechts, ausgehend vom Ursprung, und die y-Werte wachsen nach unten. Dies unterscheidet sich von anderen Zeichensystemen, bei denen sich der Koordinatenursprung in der linken, unteren Ecke befindet und die y-Werte nach oben wachsen.

Alle Pixel-Werte sind Integer - Sie können keine Dezimalzahlen verwenden, um etwas zwischen den Integerwerten anzuzeigen.

Abbildung 9.2 illustriert Javas Grafikkoordinatensystem mit dem Ursprung 0,0. Zwei der Punkte des Rechtecks sind bei 20,20 und 60,60.

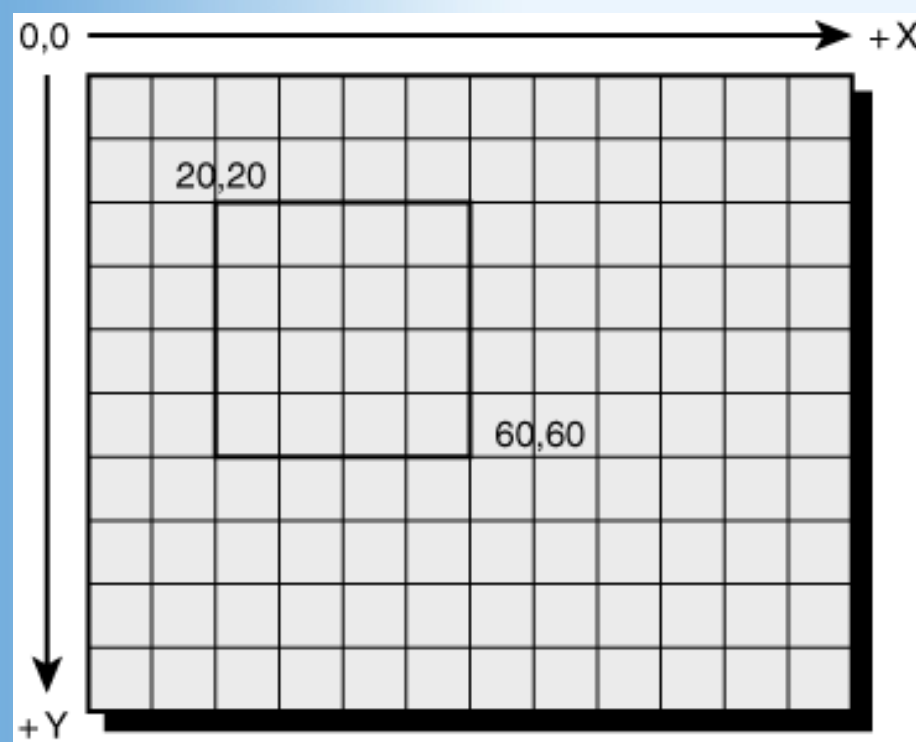


Abbildung 9.2:
Das Grafikkoordinatensystem von Java

Zeichnen und Füllen

Es stehen zwei Arten von Zeichenmethoden für viele der Figuren, die Sie in das Applet-Fenster zeichnen können: Methoden, deren Name mit `draw` beginnt und die nur den Umriß der jeweiligen Figur zeichnen, und Methoden, deren Name mit `fill` beginnt und die die jeweilige Figur mit der aktuellen Farbe füllen. Bei beiden Methodenarten wird der Umriß ebenso mit der aktuellen Farbe gezeichnet.



Sie können auch Bitmap-Dateien, wie z.B. GIF- oder JPEG-Dateien, ausgeben. Dazu verwenden Sie die Klasse Image. Darüber lernen Sie morgen mehr.

Linien

Die Methode `drawLine()` wird verwendet, um zwischen zwei Punkten eine Linie auszugeben. Die Methode erwartet vier Argumente: die x,y-Koordinaten des Startpunktes und die x,y-Koordinaten des Endpunktes:

```
drawLine(x1, y1, x2, y2);
```

Diese Methode zeichnet eine Linie von dem Punkt (x1, y1) zu dem Punkt (x2, y2). Die Breite der Linie ist mit einem Pixel festgelegt.

Fügen Sie die folgende Anweisung in die `paint()`-Methode des Map-Applets ein:

```
screen.drawLine(185, 80, 222, 80);
```

Dies zeichnet eine Linie von 185,80 nach 222,80 - die Linie unterstreicht das Wort Florida im Applet-Fenster, wie das in Abbildung 9.3 zu sehen ist.

Abbildung 9.3:
Einfügen einer Linie in ein Applet



Um ein Schleudertrauma zu verhindern, das vom häufigen Hin- und Herspringen zwischen dem Text und Ihrem Java-Quellcode-Editor resultieren kann, ist der gesamte Quelltext am Ende dieses Abschnitts abgedruckt. Bis dahin können Sie sich auf den Text konzentrieren und später den gesamten Java-Code in einem Stück eingeben.

Rechtecke

Es gibt Graphics-Methoden für zwei Arten von Rechtecken: normale Rechtecke und Rechtecke mit abgerundeten Ecken (wie die Ecken der Tasten auf den meisten Computer-Tastaturen).

Um ein normales Rechteck zu zeichnen, verwenden Sie die Methode `drawRect()` (für den Umriß) und `fillRect()` (für ein gefülltes Rechteck).

Beide Methoden erwarten vier Argumente:

- Die x,y-Koordinaten der linken, oberen Ecke des Rechtecks
- Die Breite des Rechtecks
- Die Höhe des Rechtecks

Fügen Sie die folgende Anweisung in das Map-Applet ein:

```
screen.drawRect(2, 2, 345, 345);
```

Dies fügt den Umriß eines Rechtecks ein, der sich fast an den Ecken des Applets befindet. Würden Sie hier die

Methode `fillRect()` verwenden, würde ein ausgefülltes Rechteck gezeichnet werden, das einen Großteil der Applet-Fläche einnimmt und so den unterstrichenen Text Florida überdecken würde.

Rechtecke mit abgerundeten Ecken zeichnen Sie mit den Methoden `drawRoundedRect()` und `fillRoundedRect()`. Diese erwarten dieselben ersten vier Argumente, die auch die regulären Rechteck-Methoden erwarten. Zusätzlich sind noch zwei weitere Argumente am Ende der Parameterliste vorhanden.

Diese beiden letzten Argumente definieren die Breite und die Höhe des Bereiches, in dem die Ecken gerundet werden. Je größer dieser Bereich wird, desto runder werden die Ecken. Sie können ein Rechteck sogar wie einen Kreis oder ein Oval aussehen lassen, indem Sie diese Argumente groß genug machen.

Abbildung 9.4 zeigt einige Beispiele für Rechtecke mit abgerundeten Ecken. Ein Rechteck hat für die runden Ecken eine Breite von 30 und eine Höhe von 10. Ein anderes verwendet eine Breite von 20 und eine Höhe von 20 für die runden Ecken und sieht eher wie ein Kreis aus als wie ein Rechteck.

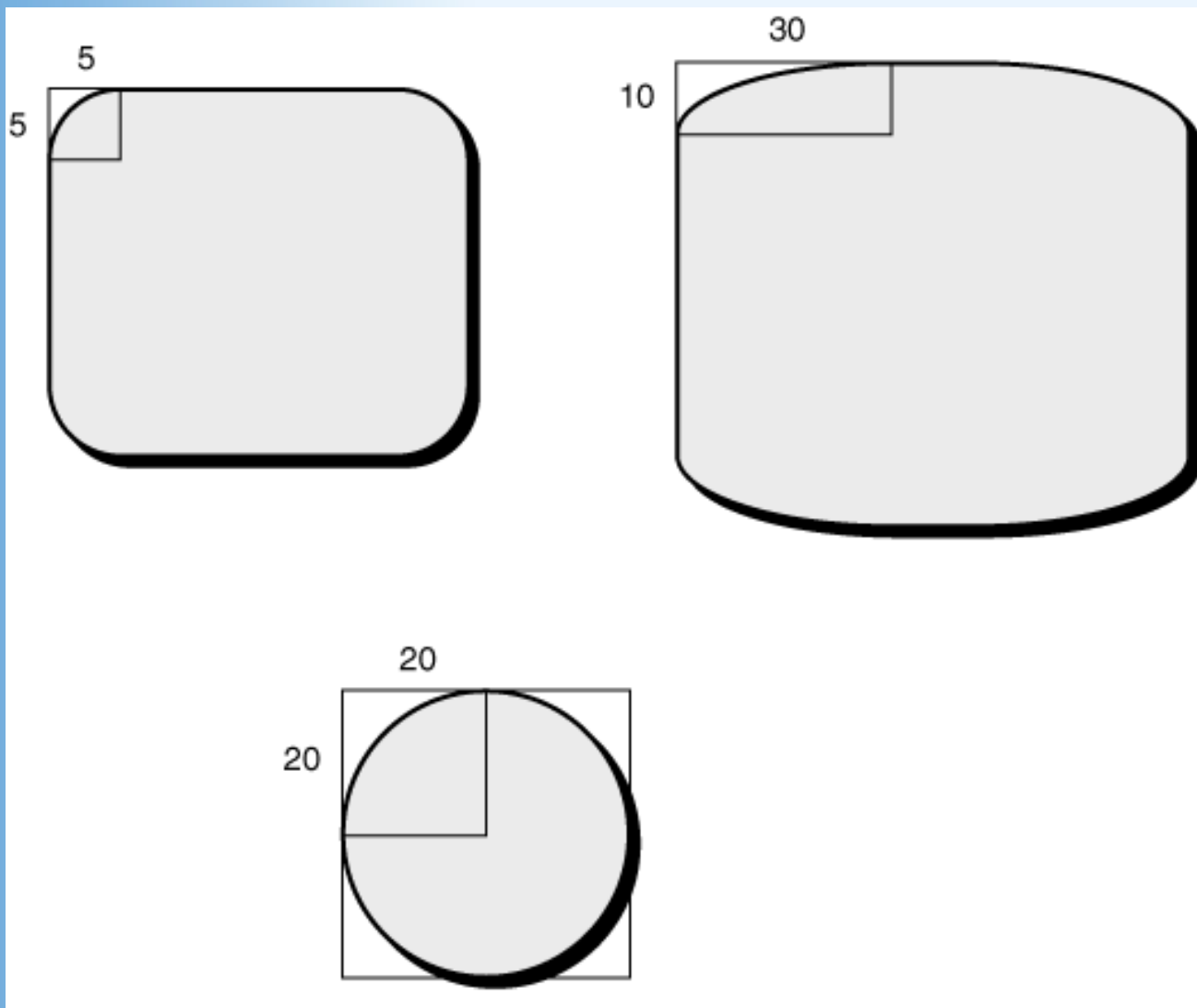


Abbildung 9.4:
Rechtecke mit abgerundeten Ecken

Fügen Sie die folgende Anweisung in die `paint()`-Methode des Map-Applets ein:

```
screen.drawRoundRect(182, 61, 43, 24, 10, 8);
```

Hiermit zeichnen Sie ein Rechteck bei den Koordinaten 182,61 mit einer Breite von 43 Pixeln und einer Höhe von 24 Pixeln. Der rechteckige Bereich der runden Ecken ist 10 Pixel breit und 8 Pixel hoch. Das Ergebnis sehen Sie in Abbildung 9.5 - eine Art Nahaufnahme eines Teils des Applets.



Abbildung 9.5: Hinzufügen eines Rechtecks in das Applet

Polygone

Polygone können mit den Methoden `drawPolygon()` und `fillPolygon()` gezeichnet werden.

Um ein Polygon zu zeichnen, müssen Sie für jeden Punkt des Polygons die x,y-Koordinaten angeben. Polygone können Sie sich als eine Reihe von Linien vorstellen, die an den Enden miteinander verbunden sind - eine Linie wird von einem Startpunkt zum Endpunkt gezeichnet. Dieser Endpunkt wird als Startpunkt für eine neue Linie verwendet und so weiter.

Sie können diese Koordinaten auf zweierlei Arten angeben:

- Als zwei Arrays mit Integer. Das eine Array nimmt alle x-Werte auf und das andere alle y-Werte.
- Als Polygon-Objekt, das mit einem Integer-Array mit x-Koordinaten und einem mit y-Koordinaten erzeugt wird.

Die zweite Methode ist wesentlich flexibler, da sie es ermöglicht, einzelne Punkte einem Polygon hinzuzufügen, bevor dieses gezeichnet wird.

Neben den x- und y-Koordinaten müssen Sie auch die Anzahl der Punkte in dem Polygon angeben. Sie können nicht mehr x,y-Koordinaten angeben, als Sie Punkte haben, bzw. Sie können auch nicht mehr Punkte angeben, als Sie Koordinatenpaare angeben. In beiden Fällen resultiert daraus ein Compiler-Fehler.

Beim Erzeugen eines Polygon-Objekts ist der erste Schritt, ein leeres Polygon mit der Anweisung `new Polygon()` wie folgt zu erstellen:

```
Polygon poly = new Polygon();
```

Als Alternative können Sie ein Polygon mit einer Reihe von Punkten, deren Koordinaten Sie in zwei Integer-Arrays angeben, erzeugen. Dafür ist es nötig, den Konstruktor `Polygon(int[], int[], int)` aufzurufen. Dabei geben Sie ein Array mit x-Werten und eines mit y-Werten und die Anzahl der Punkte an. Das folgende Beispiel verdeutlicht die Anwendung dieses Konstruktors:

```
int x[] = { 10, 20, 30, 40, 50 };
int y[] = { 15, 25, 35, 45, 55 };
int points = x.length;
Polygon poly = new Polygon(x, y, points);
```

Nachdem ein Polygon-Objekt erzeugt wurde, können Sie diesem Punkte hinzufügen. Dazu verwenden Sie die `addPoint()`-Methode des Objekts. Diese erwartet die x,y-Koordinaten des Punktes als Argument und fügt den Punkt dem Polygon hinzu. Im folgenden ein Beispiel:

```
poly.addPoint(60, 65);
```

Sobald ein Polygon-Objekt alle Punkte enthält, können Sie es mit der Methode `drawPolygon()` oder `fillPolygon()` zeichnen. Diese benötigen in diesem Fall nur ein Argument - das Polygon-Objekt, wie im folgenden gezeigt:

```
screen.drawPolygon(poly);
```

Wenn Sie die Methode `drawPolygon()` unter Java 1.02 verwenden, können Sie das Polygon schließen, indem Sie als letztes x,y-Koordinatenpaar das erste wiederholen. Andernfalls ist das Polygon an einer Seite offen.

Die Methode `fillPolygon()` schließt das Polygon automatisch, ohne daß ein übereinstimmender Anfangs- und

Endpunkt nötig ist.



Das Verhalten von `drawPolygon()` änderte sich nach der Version 1.02 von Java. Bei der Version 1.1 und 1.2 schließt `drawPolygon()` automatisch ein Polygon, wie das bei `fillPolygon()` bereits der Fall war. Wenn Sie ein Polygon mit einer offenen Ecke mit diesen Versionen erstellen wollen, verwenden Sie die Methode `drawPolyline()`. Sie arbeitet genau wie `drawPolygon()` unter Java 1.02.

Fügen Sie die folgenden Anweisungen der `paint()`-Methode des Map-Applets hinzu, um Polygone in Aktion zu sehen:

```
int x[] = { 10, 234, 253, 261, 344, 336, 295, 259, 205, 211,
           195, 191, 120, 94, 81, 12, 10 };
int y[] = { 12, 15, 25, 71, 209, 278, 310, 274, 188, 171, 174,
           118, 56, 68, 49, 37, 12 };
int pts = x.length;
Polygon poly = new Polygon(x, y, pts);
screen.drawPolygon(poly);
```

Die `Polygon`-Klasse ist Bestandteil des Paketes `java.awt`. Aus diesem Grund müssen Sie sie verfügbar machen, indem Sie die folgende Anweisung am Beginn des Map-Applets einfügen:

```
import java.awt.Polygon;
```

Abbildung 9.6 zeigt, wie das mit dem `Polygon` Map-Applet aussieht, wenn alles andere bereits gezeichnet wurde.

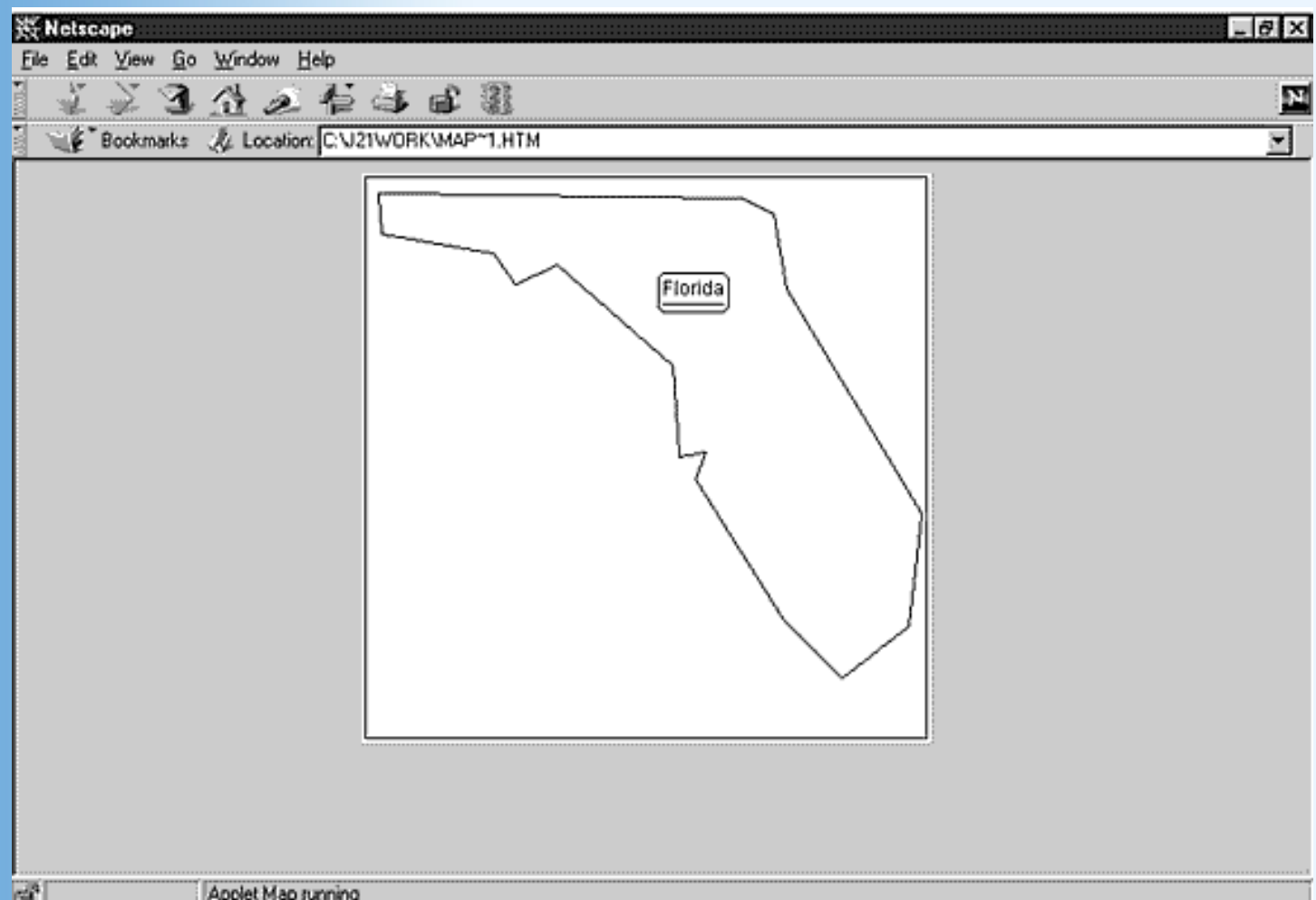


Abbildung 9.6: Hinzufügen eines Polygons in das Applet

Ovale

Die Methoden `drawOval()` und `fillOval()` werden verwendet, um Kreise und Ovale zu zeichnen.

Diese Methoden erwarten vier Argumente:

- Die x,y-Koordinaten des Ovals
- Die Breite und Höhe des Ovals, die bei Kreisen denselben Wert haben

Da ein Oval keine Ecken hat, fragen Sie sich vielleicht, auf was sich die x,y-Koordinaten beziehen. Ovale werden auf die gleiche Weise gehandhabt wie die Rechtecke mit gerundeten Ecken. Die x,y-Koordinaten stellen die linke obere Ecke des rechteckigen Bereichs dar, in dem das Oval beschrieben ist.

Kehren Sie zum Map-Applet zurück, und fügen Sie die folgenden Anweisungen hinzu:

```
screen.fillOval(235,140,15,15);
screen.fillOval(225,130,15,15);
screen.fillOval(245,130,15,15);
```

Hier werden ausschließlich fill-Methoden anstelle der draw-Methoden verwendet. Deshalb entstehen drei schwarz gefüllte Kreise, die an einem Fleck in Zentral-Florida miteinander verbunden sind, wie das in Abbildung 9.7 zu sehen ist.

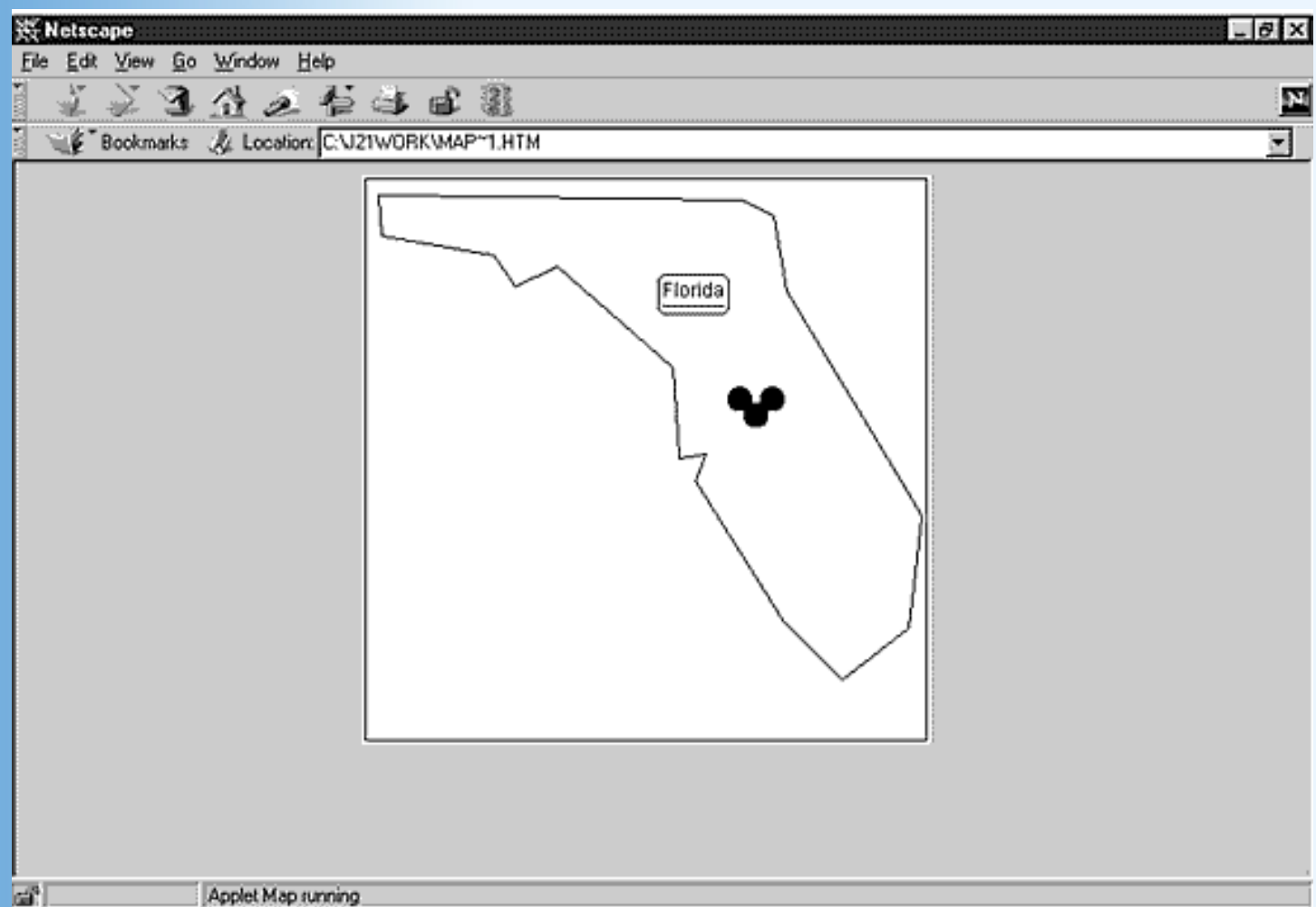


Abbildung 9.7:
Hinzufügen
dreier Kreise in das Applet

Bögen

Unter allen Zeichenmethoden sind Bögen die komplexesten bei der Konstruktion. Ein Bogen ist ein Teil eines Ovals und in Java als Bogen implementiert, der nur teilweise gezeichnet wird.

Bögen werden mit den Methoden `drawArc()` und `fillArc()` gezeichnet. Diese Methoden erwarten sechs Argumente:

- Die x,y-Koordinaten des Ovals
- Die Breite und Höhe des Ovals
- Den Winkel, bei dem der Bogen beginnt
- Den Winkel den der Bogen überstreicht

Die ersten vier Argumente entsprechen denen für ein Oval und funktionieren auf dieselbe Art und Weise.

Der Startwinkel des Bogens reicht von 0 bis 359 Grad und wird gegen den Uhrzeigersinn gezählt. Auf einem Ziffernblatt einer Uhr würde 0 3 Uhr, 90 12 Uhr, 180 9 Uhr und 270 6 Uhr entsprechen.

Der Winkel, den ein Bogen überstreicht, reicht von 0 bis 359 Grad gegen den Uhrzeigersinn und von 0 bis -359 Grad im Uhrzeigersinn.

Abbildung 9.8 zeigt, wie die letzten beiden Argumente berechnet werden.

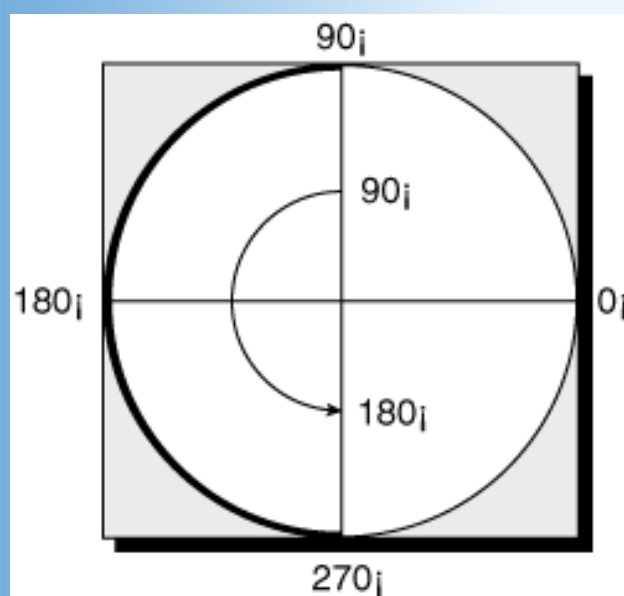


Abbildung 9.8:

Die Abmessungen eines Bogens ermitteln

Gefüllte Bögen werden so gezeichnet, als wären sie Stücke einer Torte. Anstatt die beiden Endpunkte miteinander zu verbinden, werden beide Endpunkte mit dem Mittelpunkt des Ovals des Bogens verbunden.

Im folgenden sehen Sie einen Beispiel für einen Aufruf der Methode `drawArc()`:

```
screen.drawArc(20, 25, 315, 150, 5, -190);
```

Diese Anweisung zeichnet einen Bogen eines Ovals mit den Koordinaten 20,25, einer Breite von 315 Pixeln und einer Höhe von 150 Pixeln. Der Bogen beginnt bei der 5- Grad-Marke und überstreicht einen Winkel von 190 Grad im Uhrzeigersinn. Der Bogen wird in Abbildung 9.9 gezeigt.

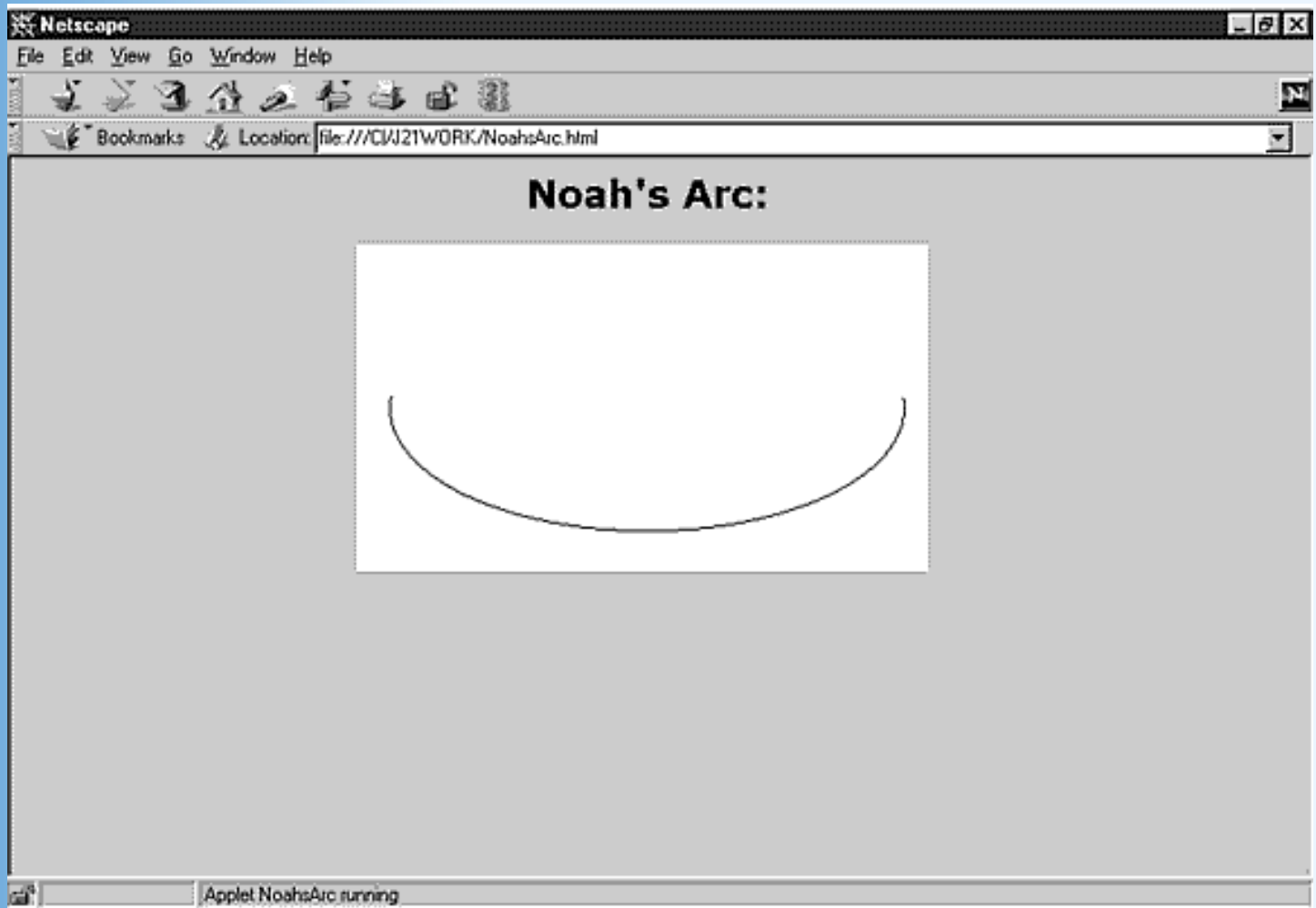


Abbildung 9.9:
Ein Bogen

Als letzte Zutat für das Map-Applet werden wir eine Reihe kleiner Bögen zeichnen, die vier Argumente gemeinsam haben:

- Das Oval eines jeden Bogens hat eine Breite und Höhe von 10 Pixeln, was das Oval zu einem Kreis werden läßt.
- Jeder Bogen beginnt bei 0 Grad und überstreicht einen Winkel von 180 Grad. D.h., die Bögen sind Halbkreise.

Die x,y-Koordinaten der Bögen ändern sich. Zwei for-Schleifen gehen dabei eine Reihe von x- und y-Werten durch.

Fügen Sie die beiden folgenden Anweisungen in die paint()-Methode des Map-Applets ein:

```
for (int ax = 50; ax < 150; ax += 10)
    for (int ay = 120; ay < 320 ; ay += 10)
        screen.drawArc(ax, ay, 10, 10, 0, -180);
```

Die Tatsache, daß sich hier zwei for-Schleifen ineinander befinden, mag auf den ersten Blick etwas verwirrend erscheinen. Im folgenden finden Sie deshalb die ersten sechs x,y-Koordinaten, die von den Schleifen erzeugt werden:

```
50, 120
50, 130
50, 140
50, 150
50, 160
50, 170
```


Wie Sie sehen, können, ändert sich die x-Koordinate - festgelegt durch ax - nicht. Diese ändert sich so lange nicht, bis die gesamte ay-Schleife durchgelaufen ist. Wenn dies geschehen ist, wird der Wert von ax um 10 erhöht und die ay-Schleife erneut komplett ausgeführt.

Kompilieren Sie das Map-Applet, um zu sehen, welchen Effekt diese Schleifen durch das Zeichnen einer ganzen Reihe kleiner Halbkreise produzieren. Listing 9.3 enthält den gesamten und endgültigen Quelltext für Map.java - inklusive aller Zeichenanweisungen, die in diesem Abschnitt behandelt wurden.

Listing 9.3: Der gesamte und endgültige Quelltext von Map.java

```

1: import java.awt.Graphics;
2: import java.awt.Polygon;
3:
4: public class Map extends java.applet.Applet {
5:     public void paint(Graphics screen) {
6:         screen.drawString("Florida", 185, 75);
7:         screen.drawLine(185,80,222,80);
8:         screen.drawRect(2, 2, 345, 345);
9:         screen.drawRoundRect(182,61,43,24,10,8);
10:        int x[] = { 10, 234, 253, 261, 344, 336, 295, 259, 205, 211,
11:                  195, 191, 120, 94, 81, 12, 10 };
12:        int y[] = { 12, 15, 25, 71, 209, 278, 310, 274, 188, 171, 174,
13:                  118, 56, 68, 49, 37, 12 };
14:        int pts = x.length;
15:        Polygon poly = new Polygon(x, y, pts);
16:        screen.drawPolygon(poly);
17:        screen.fillOval(235,140,15,15);
18:        screen.fillOval(225,130,15,15);
19:        screen.fillOval(245,130,15,15);
20:        for (int ax = 50; ax < 150; ax += 10)
21:            for (int ay = 120; ay < 320 ; ay += 10)
22:                screen.drawArc(ax, ay, 10, 10, 0, -180);
23:    }
24: }
```

Abbildung 9.10 zeigt das Map-Applet, das mit den elementaren Zeichenmethoden von Java gezeichnet wurde.

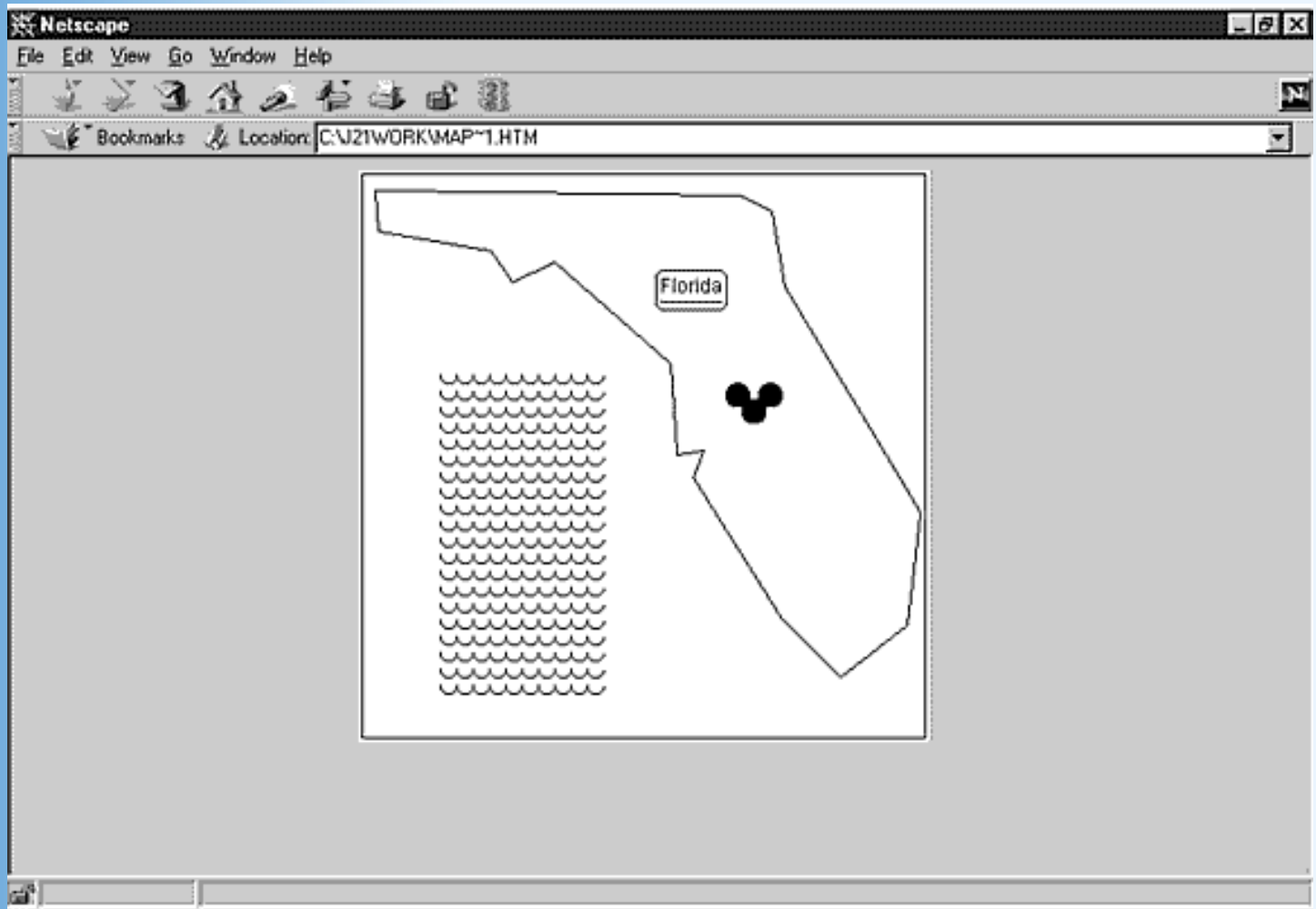


Abbildung 9.10:
Das Map-Applet

Obwohl kein Kartograph angesichts dieses Applets um die Sicherheit seines Arbeitsplatzes besorgt wäre, kombiniert dieses Applet Beispiele für die meisten Zeichen-Features, die unter Java über die Klasse Graphics zur Verfügung stehen. Ein Applet wie dieses kann über die Verwendung von Font- und Color-Objekten erweitert werden. Zusätzlich könnten die Zeichenoperationen neu arrangiert werden, um das Endprodukt zu verbessern.

Kopieren und Löschen

Die Graphics-Klasse bietet auch einiges an Cut-and-paste-Funktionalität, die das Applet-Fenster einbezieht:

- Die Methode `copyArea()` kopiert einen rechteckigen Bereich des Applet-Fensters in einen anderen Bereich des Fensters.
- Die Methode `clearRect()` löscht den Inhalt eines rechteckigen Bereichs des Applet-Fensters.

Die Methode `copyArea()` erwartet sechs Argumente:

- Die x,y-Koordinaten des zu kopierenden Bereichs
- Die Breite und Höhe dieses Bereichs
- Die horizontale und vertikale Distanz, die zwischen dem zu kopierenden Bereich und dem Bereich liegt, in dem die Kopie angezeigt werden soll.

Die folgende Anweisung kopiert einen Bereich von 100 auf 100 Pixel in einen Bereich, der 50 Pixel rechts und 25 unterhalb davon liegt:

```
screen.copyArea(0, 0, 100, 100, 50, 25);
```

Die Methode `clearRect()` erwartet dieselben vier Argumente wie die Methoden `drawRect()` und `fillRect()`. Die

Methode füllt den angegebenen Bereich mit der aktuellen Hintergrundfarbe des Applets. Sie lernen später am heutigen Tag, wie Sie die Hintergrundfarbe festlegen.

Sie können die Größe des Fensters über die Methode `size()` ermitteln, wenn Sie das gesamte Applet-Fenster löschen wollen. Diese Methode gibt ein Dimensions-Objekt zurück. Die Variablen `width` und `height` dieses Objekts repräsentieren die Dimensionen des Applets. Die folgende Anweisung ist ein Beispiel für dieses Vorgehen:

```
screen.clearRect(0, 0, size().width, size().height);
```



Die `size()`-Methode wurde in den Java-Versionen nach 1.02 umbenannt. Sie funktioniert unter Java 1.2 weiterhin. Der Compiler gibt allerdings eine Warnung aus, daß diese Methode verworfen wurde. D.h. es ist eine neuere Methode verfügbar, die diese Methode ersetzt. Die Methode `getSize()` von Java 1.2 funktioniert genauso wie die `size()`-Methode. Die Änderung des Namens ist Teil der Anstrengungen von JavaSoft, eine konsistente Benennung der Methoden in der gesamten Klassenbibliothek von Java zu erreichen.

Text und Schriften

Die Objekte der Klasse `java.awt.Font` werden verwendet, um verschiedene Schriften für die Methode `drawString()` zur Verfügung zu haben. Font-Objekte repräsentieren den Namen, den Stil und die Punktgröße einer Schrift. Eine andere Klasse, `FontMetrics`, bietet Methoden, um die Größe der angezeigten Zeichen in der festgelegten Schrift zu ermitteln. Dies kann zur Formatierung und Zentrierung von Text verwendet werden.

Font-Objekte erzeugen

Ein Font-Objekt wird erzeugt, indem man den Konstruktor der Klasse mit drei Argumenten aufruft:

- Den Namen der Schrift
- Den Stil der Schrift
- Die Größe der Schrift in Punkt

Der Name der Schrift kann ein bestimmter Schriftname, wie z.B. Arial oder Garamond Old Style sein. Diese Schrift wird verwendet, wenn die Schrift auf dem System, auf dem das Java-Programm ausgeführt wird, vorhanden ist.

Sie können auch die Namen der in Java integrierten Schriften verwenden: TimesRoman, Helvetica, Courier, Dialog und DialogInput.



In Java 1.1 und höher sollten die Schriftnamen TimesRoman, Helvetica und Courier durch `serif`, `sanserif` und `monospaced` ersetzt werden. Diese generischen Namen geben den Stil der Schrift an, ohne eine bestimmte Schriftfamilie festzulegen, die diesen repräsentiert. Dies stellt eine bessere Wahl dar, da manche Schriftfamilien nicht unter allen Java -Implementationen vorhanden sind. Auf diese Weise kann die Schrift ausgewählt werden, die auf der Plattform dem gewünschten Schriftstil (wie z.B. `serif`) am nächsten kommt.



Falls Sie das JDK unter Solaris 2.6 einsetzen, sollten Sie das SUNWi1of Paket für zusätzliche Latin-1 Schriften laden, da es ansonsten zu Warnungen beim Übersetzen kommen kann.

Es können drei verschiedene Schriftstile über die Konstanten `Font.PLAIN`, `Font.BOLD` und `Font.ITALIC` ausgewählt werden. Diese Konstanten sind Integer und können, um die Effekte zu kombinieren, addiert werden.

Das letzte Argument des `Font()`-Konstruktors ist die Größe der Schrift in Punkt.

Die folgende Anweisung erzeugt ein Font-Objekt der Schrift Dialog in 24-Punkt, fett und kursiv:

```
Font f = new Font("Dialog", Font.BOLD + Font.ITALIC, 24);
```

Zeichen und Strings ausgeben

Um die aktuelle Schrift festzulegen, wird die Methode `setFont()` der `Graphics`-Klasse verwendet. Als Argument wird dieser Methode ein Font-Objekt übergeben. Die folgende Anweisung verwendet ein Font-Objekt mit dem Namen `ft`:

```
screen.setFont(ft);
```

Text kann in einem Applet-Fenster mit der Methode `drawString()` ausgegeben werden. Diese Methode verwendet die aktuell ausgewählte Schrift. Wenn keine Schrift ausgewählt wurde, verwendet sie die Standardschrift. Mit der Methode `setFont()` können Sie jederzeit eine Schrift zur aktuellen Schrift machen.

Die folgende `paint()`-Methode erzeugt ein neues Font-Objekt, setzt dieses Objekt als die aktuelle Schrift und gibt den String "I'm very font of you." bei den Koordinaten 10,100 aus.

```
public void paint(Graphics screen) {
    Font f = new Font("TimesRoman", Font.PLAIN, 72);
    screen.setFont(f);
    screen.drawString("I'm very font of you.", 10, 100);
}
```

Informationen über Schriften ermitteln

Die Klasse `FontMetrics` kann zur Ermittlung detaillierter Informationen über eine Schrift, z.B. die Breite oder Höhe von Zeichen, die damit angezeigt werden können, verwendet werden.

Um die Methoden dieser Klasse zu verwenden, muß ein `FontMetrics`-Objekt mit der Methode `getFontMetrics()` erzeugt werden. Die Methode erwartet nur ein einziges Argument: ein Font-Objekt.

Tabelle 9.1 führt einige der Informationen auf, die Sie mit Hilfe der `FontMetrics`- Klasse ermitteln können. Alle diese Methoden sollten über ein `FontMetrics`-Objekt aufgerufen werden.

Methodenname	Beschreibung
<code>stringWidth(String)</code>	Gibt die gesamte Breite des übergebenen Strings in Pixeln zurück.
<code>charWidth(char)</code>	Gibt die Breite des übergebenen Zeichens zurück.
<code>getHeight()</code>	Gibt die Gesamthöhe der Schrift zurück.

Tabelle 9.1: Methoden der Klasse `FontMetrics`

Listing 9.4 zeigt, wie die Klassen `Font` und `FontMetrics` verwendet werden können. Das `SoLong`-Applet zeigt einen String in der Mitte des Applet-Fensters an. Mit Hilfe der `FontMetrics`-Klasse wird dazu die Breite des Strings in der aktuellen Schrift ermittelt.

Listing 9.4: Der gesamte Quelltext von `SoLong.java`

```

1: import java.awt.Font;
2: import java.awt.Graphics;
3: import java.awt.FontMetrics;
4:
5: public class SoLong extends java.applet.Applet {
6:
7:     public void paint(Graphics screen) {
8:         Font f = new Font("Courier", Font.BOLD, 18);
9:         FontMetrics fm = getFontMetrics(f);
10:        screen.setFont(f);
11:        String s = "So long, and thanks for all the fish.";
12:        int x = (size().width - fm.stringWidth(s)) / 2;
13:        int y = size().height / 2;
14:        screen.drawString(s, x, y);
15:    }
16: }

```

Abbildung 9.11 zeigt zwei Kopien des SoLong-Applets auf einer Webseite. Jede weist eine andere Breite des Applet-Fensters auf.

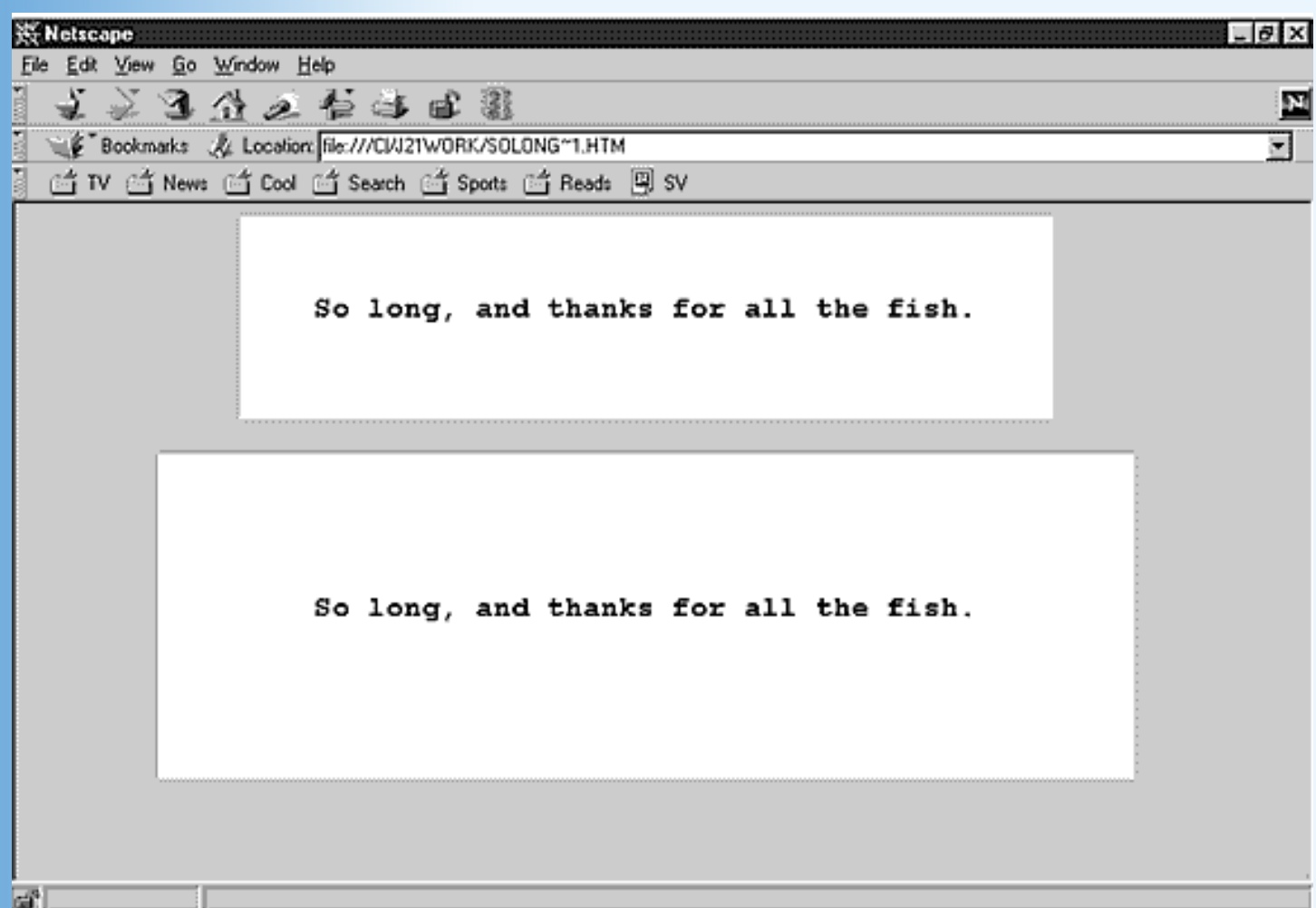


Abbildung 9.11:
Zwei Kopien des SoLong- Applets

Die `size()`-Methode in den Zeilen 12 und 13 sollte durch die `getSize()`-Methode ersetzt werden, wenn Sie ein Applet für die Java-Version 1.1 oder höher schreiben. Die Größe des Applet-Fensters innerhalb des Applets zu ermitteln ist der Festlegung der exakten Größe im Applet vorzuziehen, da dies anpassungsfähiger ist. Sie können den HTML-Code der Webseite ändern, ohne das Programm ändern zu müssen, und es wird dennoch

funktionieren.

Farbe

Die Klassen `Color` und `ColorSpace` des Paketes `java.awt` können Sie verwenden, um Ihre Applets bunter zu machen. Mit diesen Klassen können Sie die aktuelle Farbe für Zeichenoperationen sowie die Hintergrundfarbe für ein Applet und andere Fenster setzen. Sie haben auch die Möglichkeit, eine Farbe von einem Farbbeschreibungssystem in ein anderes zu konvertieren.

Standardmäßig verwendet Java Farben, die nach dem Farbbeschreibungssystem sRGB definiert sind. In diesem System wird eine Farbe über die Anteile der Farben Rot, Grün und Blau, die in ihr enthalten sind, definiert - hier kommt das R, G und B ins Spiel. Jede der drei Komponenten kann durch einen Integer-Wert zwischen 0 und 255 repräsentiert werden. Schwarz hat dann die Anteile 0,0,0 - sprich: es ist weder Rot noch Grün noch Blau vorhanden. Weiß dagegen hat die Anteile 255,255,255 - der Maximalwert aller drei Komponenten. sRGB-Werte lassen sich auch mit drei Fließkommazahlen darstellen, die jeweils einen Wert zwischen 0.0 und 1.0 haben. Java kann Millionen von Farben zwischen den beiden Extremwerten über sRGB erzeugen.

Ein Farbsystem wird in Java als *Color Space* (Farbraum) bezeichnet, und sRGB ist nur ein solcher Farbraum, der in einem Programm verwendet werden kann. Ein Ausgabegerät wie z.B. ein Monitor oder ein Drucker haben ihren eigenen Farbraum.

Wenn Sie etwas in einer bestimmten Farbe anzeigen oder drucken, kann es passieren, daß das Ausgabegerät diese Farbe nicht unterstützt. In diesem Fall wird entweder die Farbe durch eine andere Farbe substituiert, oder es wird ein Dither-Muster verwendet, um die nicht verfügbare Farbe anzunähern. Dies geschieht häufig im World Wide Web, wenn eine nicht verfügbare Farbe durch ein Dither-Muster aus zwei oder mehr Farben, die die fehlende Farbe annähern, ersetzt wird.

Die Realität des Farbmanagements in der Praxis sieht so aus, daß nicht alle Farben, die Sie über sRGB festlegen, auf allen Ausgabegeräten zur Verfügung stehen werden. Wenn Sie eine feinere Kontrolle über die Farbe benötigen, können Sie die Klasse `ColorSpace` und andere Klassen, die mit Java 1.2 eingeführt wurden, verwenden.

Für die meisten Programme wird das standardmäßig zur Definition von Farben verwendete sRGB völlig ausreichend sein.

Color-Objekte verwenden

Um die aktuelle Zeichenfarbe zu setzen, muß entweder ein `Color`-Objekt erzeugt werden, das die Farbe repräsentiert, oder Sie müssen eine der Standardfarben verwenden, die in der `Color`-Klasse verfügbar sind.

Es gibt zwei Möglichkeiten, den Konstruktor der Klasse `Color` aufzurufen, um eine Farbe zu erzeugen:

- Mit drei Integern, die den sRGB-Wert der gewünschten Farbe darstellen.
- Mit drei Fließkommawerten, die den gewünschten sRGB-Wert repräsentieren.

Sie können den sRGB-Wert einer Farbe entweder über drei int- oder drei float-Werte angeben. Die folgenden Anweisungen zeigen Beispiele hierfür:

```
Color c1 = new Color(0.807F, 1F, 0F);
Color c2 = new Color(255, 204, 102);
```

Das `c1`-Objekt beschreibt ein Neongrün und `c2` eine in etwa karamelfarbige Farbe.



Es passiert sehr leicht, daß man Fießkomma-Literale, wie z.B. 0F oder 1F, mit Hexadezimalzahlen verwechselt, die an Tag 3 besprochen wurden. Farben werden häufig als Hexadezimalwerte angegeben, wie das z.B. beim Festlegen der Hintergrundfarbe im <BODY>-Tag einer HTML-Seite der Fall ist. Keine der Java-Klassen und -Methoden, mit denen Sie arbeiten, erwarten hexadezimale Argumente. D.h., wenn Sie Literale wie 0F oder 1F sehen, können Sie sicher sein, daß Sie es mit Fließkommazahlen zu tun haben.

Die aktuelle Farbe ermitteln und festlegen

Die aktuelle Farbe wird über die Methode `setColor()` der Klasse `Graphics` festgelegt. Diese Methode muß über das `Graphics`-Objekt aufgerufen werden, das den Bereich repräsentiert, in den Sie zeichnen. In einem Applet ist dies das Objekt, das der `paint()`-Methode übergeben wird.

Eine Möglichkeit, eine Farbe zu setzen, ist, eine der Standardfarben zu verwenden, die als Klassenvariablen der Klasse `Color` zu Verfügung stehen.

Diese Farben verwenden die folgenden `Color`-Variablen (die sRGB-Werte sind in Klammern dahinter angegeben):

Farbe	Variable	RGB-Wert
Schwarz	<code>black</code>	(0,0,0)
Blau	<code>blue</code>	(0,0,255)
Cyan	<code>cyan</code>	(0,255,255)
Dunkelgrau	<code>darkGray</code>	(64,64,64)
Grau	<code>gray</code>	(128,128,128)
Grün	<code>green</code>	(0,255,0)
Hellgrau	<code>lightGray</code>	(192,192,192)
Magenta	<code>magenta</code>	(255,0,255)
Orange	<code>orange</code>	(255,200,0)
Rosa	<code>pink</code>	(255,175,175)
Rot	<code>red</code>	(255,0,0)
Weiß	<code>white</code>	(255,255,255)
Gelb	<code>yellow</code>	(255,255,0)

Die folgende Anweisung setzt die aktuelle Farbe für das `Screen`-Objekt mit einer der Standard-Klassenvariablen:

```
screen.setColor(Color.pink);
```

Wenn Sie ein `Color`-Objekt erzeugt haben, kann es auf ähnliche Weise als aktuelle Farbe gesetzt werden:

```
Color brush = new Color(255, 204, 102);
screen.setColor(brush);
```

Nachdem Sie die aktuelle Farbe gesetzt haben, erscheinen alle Zeichenoperationen in dieser Farbe.

Sie können die Hintergrundfarbe bzw. die Vordergrundfarbe eines Applet-Fensters über die Methoden `setBackground()` bzw. `setForeground()` setzen. Diese Methoden erbt die Applet Klasse von einer ihrer Superklassen, so daß alle Applets, die Sie erzeugen, diese Methoden erben.

Die Methode `setBackground()` legt die Farbe des Hintergrundes des Applet-Fensters fest. Sie erwartet ein einziges Argument - ein `Color`-Objekt:

```
setBackground(Color.white);
```

Die Methode `setForeground()` wird über Komponenten der Benutzerschnittstelle und nicht über ein `Graphics`-Objekt aufgerufen. Sie arbeitet genauso wie die Methode `setColor()`, nur daß sie die Farbe einer Komponente, wie z.B. eine Schaltfläche oder ein Fenster, ändert.

Da ein Applet ein Fenster ist, können Sie die `setForeground()`-Methode in der `init()`-Methode verwenden, um die Farbe für Zeichenoperationen festzulegen. Diese Farbe wird so lange verwendet, bis eine andere Farbe entweder über `setForeground()` oder über `setColor()` gewählt wird.

Wenn Sie die aktuelle Farbe ermitteln wollen, können Sie die Methode `getColor()` über ein `Graphics`-Objekt aufrufen oder die Methoden `getForeground()` bzw. `getBackground()` der Applet-Klasse verwenden.

Die folgende Anweisung setzt die aktuelle Farbe des `screen`-Objekts - ein Objekt der Klasse `Graphics` - auf die Hintergrundfarbe des Applets:

```
screen.setColor(getBackground());
```

Fortgeschrittene Grafikoperationen mit Java2D

Eine der Erweiterungen, die Java 1.2 bietet, ist Java2D. Dabei handelt es sich um einen Satz von Klassen, die Ihnen 2D-Grafiken, Bilder und Text in hoher Qualität in Ihren Programmen ermöglichen. Die Java2D-Klassen erweitern die bestehenden Möglichkeiten der vorhandenen `java.awt`-Klassen, die zur Verarbeitung von Grafik verwendet werden - darunter auch die, die Sie heute kennengelernt haben. Sie ersetzen die vorhandenen Klassen allerdings nicht - Sie können die anderen Klassen und Programme, die diese implementieren, weiterhin verwenden.

Java2D hat unter anderem die folgenden Features:

- Spezielle Füllmuster, wie z.B. Verlaufsfüllungen oder Musterfüllungen
- Möglichkeiten zur Definition der Strichstärke und des Strichstils beim Zeichnen
- Anti-Aliasing, um bei gezeichneten Objekten Treppchen-Effekte zu vermeiden

Benutzer- und Gerätekoordinatensysteme

Eines der Konzepte, das mit Java2D eingeführt wurde, ist die Unterscheidung zwischen dem Kooordinatensystem eines Ausgabegerätes und dem Koordinatensystem, auf das Sie sich beim Zeichnen eines Objekts beziehen.



Bisher wurde für alle Zeichenoperationen (dies gilt für alle Zeichenoperationen vor Java 1.2) nur das Gerätekoordinatensystem verwendet. Sie legen die x,y-Koordinaten auf einer Ausgabefläche wie z.B. einem Applet-Fenster fest. Diese Koordinaten wurden für das Zeichnen von Linien und anderen Elementen sowie die Ausgabe von Text verwendet.

Graphics2D-Objekte erzeugen

Die Zeichenoperationen, die Sie bisher erlernt haben, wurden über ein `Graphics`-Objekt aufgerufen, das den Ausgabebereich repräsentiert - z.B. ein Applet-Fenster. Für Java2D muß mit diesem Objekt ein neues

Graphics2D-Objekt erzeugt werden, wie das in der folgenden paint()-Methode der Fall ist:

```
public void paint(Graphics screen) {
    Graphics2D screen2D = (Graphics2D)screen;
}
```

Das Objekt screen2D in diesem Beispiel wurde über Casting erzeugt. Es ist das screen-Objekt, das von einem Objekt der Graphics-Klasse in ein Objekt der Klasse Graphics2D konvertiert wurde.

Alle Java2D-Zeichenoperationen müssen über ein Graphics2D-Objekt aufgerufen werden. Graphics2D ist Bestandteil des Paketes java.awt.

Festlegen der Darstellungsattribute

Der nächste Schritt beim 2D-Zeichnen ist es, festzulegen, wie ein gezeichnetes Objekt dargestellt werden soll. Objekte, die nicht 2D sind, können nur ein Attribut wählen: die Farbe. 2D bietet eine breite Palette an Attributen, um die Farbe, die Linienstärke, Füllmuster, Transparenz und vieles mehr festzulegen.

2D Farben

Farben werden mit der Methode setColor() gesetzt. Dies funktioniert genauso wie mit der Graphics-Methode gleichen Namens. Im folgenden ein Beispiel:

```
screen2D.setColor(Color.black);
```



Obwohl einige der 2D-Methoden genauso wie deren Nicht-2D-Gegenstücke verwendet werden, müssen sie über ein Graphics2D-Objekt aufgerufen werden, um die Fähigkeiten von Java2D zu verwenden.

Java2D benötigt ein zweites Koordinatensystem, das Sie bei der Erzeugung - dem eigentlichen Zeichnen - verwenden. Dies wird als *Benutzerkoordinatensystem* bezeichnet.

Bevor irgendeine Zeichenoperation stattgefunden hat, befindet sich der Ursprung (die Koordinaten 0,0) des Benutzerkoordinatensystems und der des Gerätekoordinatensystems an der gleichen Stelle - der linken, oberen Ecke des Zeichenbereiches.

Der Ursprung des Benutzerkoordinatensystems kann als Ergebnis der ausgeführten 2D-Zeichenoperationen wandern. Als Folge einer 2D-Rotation können sogar die x- und y-Achse vertauscht werden. Sie lernen mehr über die beiden verschiedenen Koordinatensysteme, während Sie mit Java2D arbeiten.

Füllmuster

Füllmusterkontrollieren, wie ein gezeichnetes Objekt gefüllt wird. Mit Java2D können Sie eine Farbe, einen Verlauf, eine Textur oder ein Muster nach Ihren eigenen Vorstellungen verwenden.

Ein Füllmuster wird über die Methode setPaint() von Graphics2D definiert. Diese erwartet ein Paint-Objekt als einziges Argument. Die Paint-Schnittstelle wird von jeder Klasse implementiert, deren Objekte als Füllmuster verwendet werden können, darunter GradientPaint, TexturePaint und Color. Letzteres könnte Sie etwas überraschen. Allerdings ist die Verwendung eines Color-Objekts zusammen mit der setPaint() -Methode dasselbe, wie ein Objekt mit einer Farbe als Muster zu füllen.



Eine *Verlaufsfüllung* ist ein abgestufter Wechsel von einer Farbe an einem Koordinatenpunkt zu einer anderen Farbe an einem anderen Koordinatenpunkt. Der Wechsel kann zwischen den Punkten nur einmal geschehen, was als *azyklischer Verlauf*, oder wiederholt, was als *zyklischer Verlauf* bezeichnet wird.

Abbildung 9.12 zeigt Beispiele für azyklische und zyklische Verläufe zwischen Weiß und einer dunkleren Farbe. Die Pfeile weisen auf die Punkte, zwischen denen die Farben wechseln.

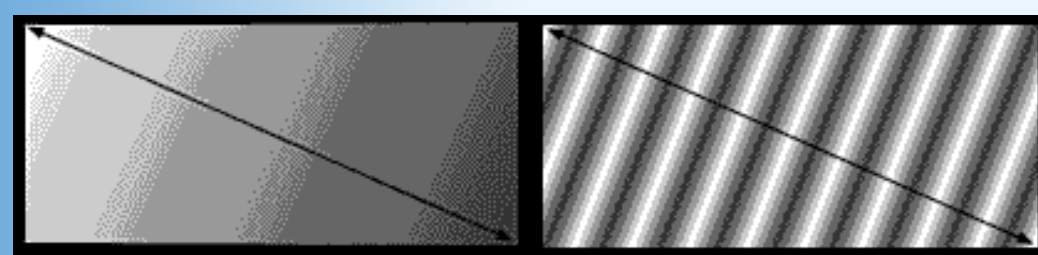


Abbildung 9.12:
Azyklische und zyklische Verläufe

Die Koordinatenpunkte in einem Verlauf beziehen sich nicht direkt auf Punkte des Graphics2D-Objekts, auf das gezeichnet wird. Statt dessen beziehen sich diese auf das Benutzerkoordinatensystem und können sogar außerhalb des Objekts, auf das gezeichnet wird, liegen.

Abbildung 9.13 illustriert dies. Beide Rechtecke in diesem Applet verwenden dasselbe GradientPaint-Objekt. Man kann sich ein Verlaufsfüllmuster als ein Stück Stoff vorstellen, das über eine ebene Oberfläche gespannt wird. Die Figures, die mit einem Verlauf gefüllt werden, sind die Schnittmuster, die aus dem Stoff ausgeschnitten werden. Und aus einem Stück Stoff kann mehr als ein Muster ausgeschnitten werden.

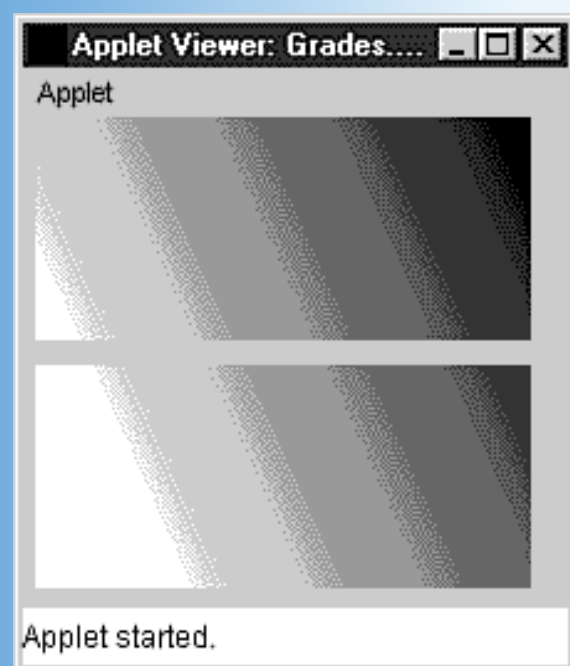


Abbildung 9.13:
Zwei Rechtecke, die dasselbe GradientPaint-Objekt verwenden

Der Aufruf des GradientPaint-Konstruktors hat das folgende Format:

```
GradientPaint(x1,y1,color1,x2,y2,color2);
```

Der Punkt x1,y1 ist der Ort, an dem der Verlauf mit der Farbe color1 startet, und am Punkt x2,y2 endet der Verlauf mit der Farbe color2.

Wenn Sie einen zyklischen Verlauf wollen, ist ein zusätzliches Argument am Ende der Argumentenliste nötig:

```
GradientPaint(x1,y1,color1,x2,y2,color2,true);
```


Das letzte Argument ist ein boolescher Wert, der für einen zyklischen Verlauf `true` sein muß. Für azyklische Verläufe ist dieses Argument `false`. Sie können es aber auch ganz weglassen - azyklische Verläufe sind das Standardverhalten.

Nachdem Sie ein `GradientPaint`-Objekt erzeugt haben, legen Sie es als das aktuelle `paint`-Attribut über die Methode `setPaint()` fest. Die folgenden Anweisungen erzeugen und wählen einen Verlauf:

```
GradientPaint pat = new GradientPaint(0f,0f,Color.white,
    100f,45f,Color.blue);
screen2D.setPaint(pat);
```

Alle folgenden Zeichenoperationen, die auf das `screen2D`-Objekt angewendet werden, verwenden dieses Füllmuster, bis ein anderes festgelegt wird.

Strichstärke und Strichstil festlegen

Wie Sie bereits gelernt haben, haben die Linien aller Nicht-2D-Zeichenoperationen eine Stärke von einem Pixel. Java2D fügt die Möglichkeit hinzu, die Stärke der Zeichenlinie zu variieren. Dazu verwenden Sie die Methode `setStroke()` mit einem `BasicStroke`-Objekt als Argument.

Ein einfacher `BasicStroke`-Konstruktor erwartet drei Argumente:

- Einen `float`-Wert, der die Linienstärke angibt - 1.0 ist der Standardwert
- Einen `int`-Wert, der die Art des Linienendes festlegt
- Einen `int`-Wert, der den Stil des Verbindungsstücks zwischen zwei Liniensegmenten festlegt

Für die Argumente für den Stil des Linienendes und der Verbindungsstücke werden Variablen der Klasse `BasicStroke` verwendet. Die Einstellung für den Stil des Linienendes bezieht sich auf Linienenden, die nicht mit anderen Linien verbunden sind. Der Stil der Verbindungsstücke wird dagegen auf Linienenden angewendet, die mit anderen Linien verbunden sind.

Mögliche Stile für Linienenden sind `CAP_BUTT`, wenn keine Abschlußpunkte verwendet werden sollen, `CAP_ROUND`, wenn an beiden Enden Kreise angezeigt werden sollen, und `CAP_SQUARE`, wenn Quadrate zum Einsatz kommen sollen. In Abbildung 9.14 sind die einzelnen Stile für die Linienenden dargestellt. Wie Sie sehen können, ist der einzige sichtbare Unterschied zwischen den Stilen `CAP_BUTT` und `CAP_SQUARE` der, daß die Linie bei `CAP_SQUARE` aufgrund des Linienendes länger ist.

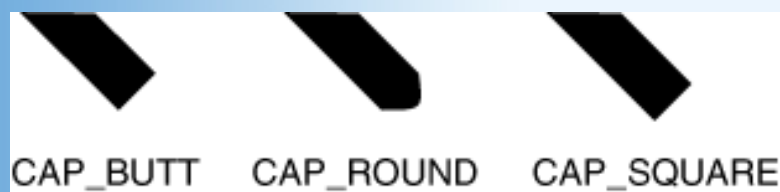


Abbildung 9.14:
Stile der Linienenden

Die möglichen Stile für die Verbindungsstücke sind `JOIN_MITER`, um Segmente zu verbinden, indem deren äußere Ecken erweitert werden, `JOIN_ROUND`, um die Ecke zwischen zwei Segmenten abzurunden, und `JOIN_BEVEL`, um die Segmente mit einer geraden Linie zu verbinden. Abbildung 9.15 zeigt Beispiele für jeden dieser Verbindungsstile.



Abbildung 9.15:
Verbindungsstile für Liniensegmente

Die folgenden Anweisungen erzeugen ein `BasicStroke`-Objekt und setzen es als aktuelles Linienattribut:

```
BasicStroke pen = BasicStroke(2.0f,
    BasicStroke.CAP_BUTT,
    BasicStroke.JOIN_ROUND);
screen2D.setStroke(pen);
```

Die Linie hat eine Breite von zwei Pixeln, keine Abschlußpunkte und abgerundete Verbindungsstücke zwischen den Segmenten.

Objekte erzeugen

Nachdem Sie ein `Graphics2D`-Objekt erzeugt und die Darstellungsattribute festgelegt haben, sind die letzten zwei Schritte, ein Objekt zu erstellen und dieses zu zeichnen.

Gezeichnete Objekte werden in Java2D erzeugt, indem man eine geometrische Form mit den Klassen des Paketes `java.awt.geom` erstellt. Sie können all das, was Sie bereits zuvor in diesem Kapitel erstellt haben, inklusive Linien, Rechtecke, Ellipsen (Ovale), Bögen und Polygone, zeichnen.

Die Klasse `Graphics2D` verfügt nicht über unterschiedliche Methoden für die einzelnen Formen, die Sie zeichnen können. Statt dessen definieren Sie die Form und verwenden als Argument für die Methode `draw()` oder `fill()`.

Linien

Linien werden mit der Klasse `Line2D.Float` erzeugt. Diese Klasse benötigt vier Argumente: die x,y-Koordinaten des einen Endpunktes, gefolgt von den x,y-Koordinaten des anderen Endpunktes. Im Anschluß ein Beispiel:

```
Line2D.Float ln = new Line2D.Float(60F, 5F, 13F, 28F);
```

Diese Anweisung erzeugt eine Linie zwischen den Punkten 60,5 und 13,28. Beachten Sie, daß das F zu den Literalen gehört, die als Argumente übergeben werden - andernfalls würde der Compiler annehmen, daß es sich um Integer handelt.

Rechtecke

Rechtecke werden mit der Klasse `Rectangle2D.Float` oder `Rectangle2D.Double` erzeugt. Der Unterschied zwischen diesen beiden Klassen ist, daß die eine float-Argumente und die andere double-Argumente benötigt.

`Rectangle2D.Float` erwartet vier Argumente: die x-Koordinate, die y-Koordinate, die Breite und die Höhe. Im folgenden ein Beispiel:

```
Rectangle2D.Float rc = new Rectangle2D.Float(10F, 13F, 40F, 20F);
```

Dies erzeugt ein Rechteck bei 10,13, das eine Breite von 40 Pixeln und eine Höhe von 20 Pixeln hat.

Ellipsen

Ovale Objekte werden in Java2D als Ellipsen bezeichnet und mit der Klasse `Ellipse2d.Float` erstellt. Dafür sind vier Argumente nötig: die x-Koordinate, die y-Koordinate, die Breite und die Höhe.

Die folgende Anweisung erzeugt eine Ellipse bei 113,25 mit einer Breite von 22 Pixeln und einer Höhe von 40 Pixeln:

```
Ellipse2D.Float ee = new Ellipse2D.Float(113,25,22,40);
```

Bögen

Bögen werden mit der Klasse `Arc2D.Float` erzeugt. Dies läuft ganz ähnlich wie bei den Nicht-2D-Gegenständen, nur daß es hier ein zusätzliches Feature gibt: Sie können angeben, wie der Bogen geschlossen wird.

`Arc2D.Float` erwartet sieben Argumente. Die ersten vier definieren die Ellipse, von der der Bogen ein Teil ist: die x-Koordinate, die y-Koordinate, die Breite und die Höhe. Die letzten drei Argumente sind der Startwinkel, die vom Bogen überstrichene Gradzahl und ein Integer, der festlegt, wie der Bogen geschlossen wird.

Der vom Bogen überstrichene Winkel wird im Uhrzeigersinn mit positiven Zahlen angegeben. Dies steht im Gegensatz zu der Art, wie Nicht-2D-Bögen verarbeitet werden.

Das letzte Argument verwendet eine von drei Klassenvariablen: `Arc2D.OPEN` für einen nicht geschlossenen Bogen, `Arc2D.CHORD`, um die Endpunkte des Bogens mit einer geraden Linie zu verbinden, und `Arc2D.PIE`, um die Endpunkte des Bogens mit dem Mittelpunkt der Ellipse zu verbinden, wie das bei einem Tortenstück der Fall ist. In Abbildung 9.16 sind die verschiedenen Stile gezeigt.



Abbildung 9.16:
Verschiedene Stile zum Schließen von Bögen



Der Stil `Arc2D.OPEN` läßt sich nicht auf gefüllte Bögen anwenden. Ein gefüllter Bogen, der den Stil `Arc2D.OPEN` verwendet, wird mit dem Stil `Arc2D.CHORD` geschlossen.

Die folgende Anweisung erzeugt ein `Arc2D.Float`-Objekt:

```
Arc2D.Float = new Arc2D.Float(27,22,42,30,33,90,Arc2D.PIE);
```

Dies erzeugt einen Bogen für ein Oval bei 27,22 mit einer Breite von 42 Pixeln und einer Höhe von 30 Pixeln. Der Bogen beginnt bei 33 Grad, überstreicht 90 Grad und wird wie ein Tortenstück geschlossen.

Polygone

Polygone werden unter Java2D erzeugt, indem man jeden einzelnen Schritt von einem Punkt eines Polygons zum nächsten definiert. Ein Polygon kann aus geraden Linien, quadratischen Kurven und Bézier-Kurven geformt werden.

Die einzelnen Schritte für die Erzeugung eines Polygons werden als `GeneralPath`-Objekt erstellt. Diese Klasse ist ebenfalls Bestandteil des Paketes `java.awt.geom`.

Ein `GeneralPath`-Objekt kann ohne Argumente erzeugt werden, wie das im folgenden gezeigt wird:

```
GeneralPath polly = new GeneralPath();
```

Die Methode `moveTo()` der Klasse `GeneralPath` wird zur Erzeugung des ersten Punktes des Polygons verwendet. Die folgende Anweisung würde verwendet werden, wenn Sie das Polygon bei dem Punkt 5,0 beginnen lassen wollten:

```
polly.moveTo(5f, 0f);
```

Nachdem Sie den ersten Punkt erzeugt haben, verwenden Sie die Methode `lineTo()`, um Linien zu erzeugen, die bei einem neuen Punkt enden. Diese Methode benötigt zwei Argumente: die x- und y-Koordinate des neuen Punktes.

Die folgenden Anweisungen fügen dem `polly`-Objekt drei neue Linien hinzu:

```
polly.lineTo(205f, 0f);
polly.lineTo(205f, 90f);
polly.lineTo(5f, 90f);
```

Die Methoden `lineTo()` und `moveTo()` benötigen float-Argumente für die Koordinatenangabe.

Wenn Sie ein Polygon schließen wollen, verwenden Sie die Methode `closePath()`. Diese wird ohne Argumente aufgerufen, wie im folgenden gezeigt:

```
polly.closePath();
```

Diese Methode schließt ein Polygon, indem sie den aktuellen Punkt mit dem Punkt, der bei dem letzten Aufruf der `moveTo()`-Methode angegeben wurde, verbindet. Sie können ein Polygon auch ohne Aufruf dieser Methode schließen, indem Sie mit der `lineTo()`-Methode eine Linie zum Ausgangspunkt ziehen.

Sobald Sie ein offenes oder ein geschlossenes Polygon erzeugt haben, können Sie es wie jede andere Figur mit der `draw()`- oder der `fill()`-Methode zeichnen. Das `polly`-Objekt ist ein Rechteck mit den Punkten (5,0), (205,0), (205,90) und (5,90).

Objekte zeichnen

Nachdem Sie die Darstellungsattribute, wie z.B. Farbe und Strichstärke, festgelegt und ein Objekt, das gezeichnet werden soll, erstellt haben, sind Sie bereit, etwas in aller 2D-Pracht zu zeichnen.

Alle gezeichneten Objekte verwenden dieselbe Methode der Klasse `Graphics2D`: `draw()` für Umrisse und `fill()` für gefüllte Objekte. Beide erwarten als einziges Argument ein Objekt.

Strings werden unter Java2D mit der Methode `drawString()` ausgegeben. Diese erwartete drei Argumente: das auszugebende String-Objekt und dessen x,y-Koordinaten. Wie auch alle anderen Koordinaten in Java2D müssen hier Fließkommazahlen anstelle von Integern angegeben werden.

Ein Java2D-Beispiel

Etwas früher am heutigen Tag haben Sie ein Karte von Florida mit den Zeichenmethoden, die in der `Graphics`-Klasse zur Verfügung stehen, erstellt. Das nächste Applet, das Sie erzeugen werden, erstellt eine überarbeitete Version dieser Karte, die 2D-Zeichentechniken verwendet.

Da alle Klassen von Java2D in der Version 1.2 von Java neu eingeführt wurden, kann dieses Applet nur mit einem Web Browser angezeigt werden, der Java 1.2 unterstützt. Beim Schreiben des Buches ist der Applet-Viewer, der dem JDK 1.2 beiliegt, die einzige Möglichkeit, dieses Applet anzuzeigen.

Listing 9.5 beinhaltet das `Map2D`-Applet. Es ist ein längeres Programm, als das bei vielen anderen Programmen in diesem Buch der Fall ist, da 2D mehr Anweisungen benötigt, um Zeichenoperationen

umzusetzen.

Listing 9.5: Der gesamte Quelltext von Map2D.java

```

1: import java.awt.*;
2: import java.awt.geom.*;
3:
4: public class Map2D extends java.applet.Applet {
5:     public void paint(Graphics screen) {
6:         Graphics2D screen2D = (Graphics2D)screen;
7:         setBackground(Color.blue);
8:         // Zeichne Wellen
9:         screen2D.setColor(Color.white);
10:        BasicStroke pen = new BasicStroke(2F,
11:            BasicStroke.CAP_BUTT, BasicStroke.JOIN_ROUND);
12:        screen2D.setStroke(pen);
13:        for (int ax = 10; ax < 340; ax += 10)
14:            for (int ay = 30; ay < 340 ; ay += 10) {
15:                Arc2D.Float wave = new Arc2D.Float(ax, ay,
16:                    10, 10, 0, 180, Arc2D.OPEN);
17:                screen2D.draw(wave);
18:            }
19:        // Zeichne Florida
20:        GradientPaint gp = new GradientPaint(0F,0F,Color.green,
21:            50F,50F,Color.orange,true);
22:        screen2D.setPaint(gp);
23:        GeneralPath fl = new GeneralPath();
24:        fl.moveTo(10F,12F);
25:        fl.lineTo(234F,15F);
26:        fl.lineTo(253F,25F);
27:        fl.lineTo(261F,71F);
28:        fl.lineTo(344F,209F);
29:        fl.lineTo(336F,278F);
30:        fl.lineTo(295F,310F);
31:        fl.lineTo(259F,274F);
32:        fl.lineTo(205F,188F);
33:        fl.lineTo(211F,171F);
34:        fl.lineTo(195F,174F);
35:        fl.lineTo(191F,118F);
36:        fl.lineTo(120F,56F);
37:        fl.lineTo(94F,68F);
38:        fl.lineTo(81F,49F);
39:        fl.lineTo(12F,37F);
40:        fl.closePath();
41:        screen2D.fill(fl);
42:        // Zeichne Ovale
43:        screen2D.setColor(Color.black);
44:        BasicStroke pen2 = new BasicStroke();
45:        screen2D.setStroke(pen2);
46:        Ellipse2D.Float e1 = new Ellipse2D.Float(235,140,15,15);
47:        Ellipse2D.Float e2 = new Ellipse2D.Float(225,130,15,15);
48:        Ellipse2D.Float e3 = new Ellipse2D.Float(245,130,15,15);
49:        screen2D.fill(e1);
50:        screen2D.fill(e2);

```



```

51:         screen2D.fill(e3);
52:     }
53: }

```

Um dieses Applet anzuzeigen, müssen Sie eine kurze HTML-Datei dafür erstellen (siehe Listing 9.6). Da dieses Applet Java-1.2-Klassen und -Methoden verwendet, kann es nur mit einem Browser angezeigt werden, der diese Version der Sprache unterstützt. Um sicherzugehen, sollten Sie den Applet-Viewer des JDK 1.2 verwenden. Der Applet-Viewer verarbeitet das `<APPLET>`-Tag und ignoriert alle anderen HTML-Tags. Es besteht also kein Grund, eine komplexe HTML-Seite zu erstellen, wenn Sie etwas mit diesem Tool anzeigen wollen.

Listing 9.6: Der gesamte Quelltext von Map2D.html

```

1: <applet code="Map2D.class" height=370 width=350>
2: </applet>

```

Einige Beobachtungen zu dem Map2D-Applet:

- In Zeile 2 werden die Klassen des Paketes `java.awt.geom` importiert. Diese Anweisung ist nötig, da die Anweisung `import java.awt.*;` in Zeile 1 nur die Klassen des `java.awt`-Paketes, nicht aber dessen Pakete importiert.
- In Zeile 6 wird das `screen2D`-Objekt erzeugt, das für alle Zeichenoperationen verwendet wird. Es entsteht durch Casting des `Graphics`-Objekts, das das Applet-Fenster repräsentiert.
- In den Zeilen 10-12 wird ein `BasicStroke`-Objekt erzeugt, das eine Linie mit einer Stärke von zwei Pixeln repräsentiert. Anschließend wird dieses Objekt mit der Methode `setStroke()` aus `Graphics2D` als aktuelles Linienattribut gesetzt.
- Die Zeilen 13-17 verwenden zwei verschachtelte `for`-Schleifen, die Wellen aus einzelnen Bögen erzeugen. Dieselbe Technik haben wir auch im Map-Applet verwendet. Im Map2D-Applet sind es aber mehr Bögen, die das Applet-Fenster bedecken.
- In den Zeilen 20 und 21 wird ein Verlaufsfüllmuster von der Farbe Grün bei 0,0 hin zu Orange bei 50,50 erzeugt. Das letzte Argument des Konstruktors, `true`, sorgt dafür, daß das Füllmuster so oft wiederholt wird, bis ein Objekt gefüllt ist.
- In Zeile 22 wird das aktuelle Verlaufsfüllmuster mit der Methode `setPaint()` und dem Objekt `gp`, das zuvor erzeugt wurde, gesetzt.
- In den Zeilen 23-41 wird das Polygon erzeugt, das die Form von Florida hat. Das Polygon wird mit dem Verlauf von Grün nach Orange gefüllt, da dies das aktuell gewählte Füllmuster ist.
- In Zeile 43 wird die aktuelle Farbe auf Schwarz gesetzt. Dies ersetzt den Verlauf bei der nächsten Zeichenoperation, da auch Farben Füllmuster sind.
- In der Zeile 44 wird ein neues `BasicStroke`-Objekt ohne Argumente erzeugt, woraus die Standardlinie mit einer Breite von einem Pixel resultiert.
- In Zeile 45 wird die aktuelle Linienbreite auf das neue `BasicStroke`-Objekt `pen2` gesetzt.
- In den Zeilen 46-51 werden drei Ellipsen bei den Punkten (235,140), (225,130) und (245,130) erzeugt. Jede davon ist 15 Pixel breit und 15 Pixel hoch, d.h. es sind Kreise.

Abbildung 9.17 zeigt die Ausgabe des Map2D-Applets im Applet-Viewer.

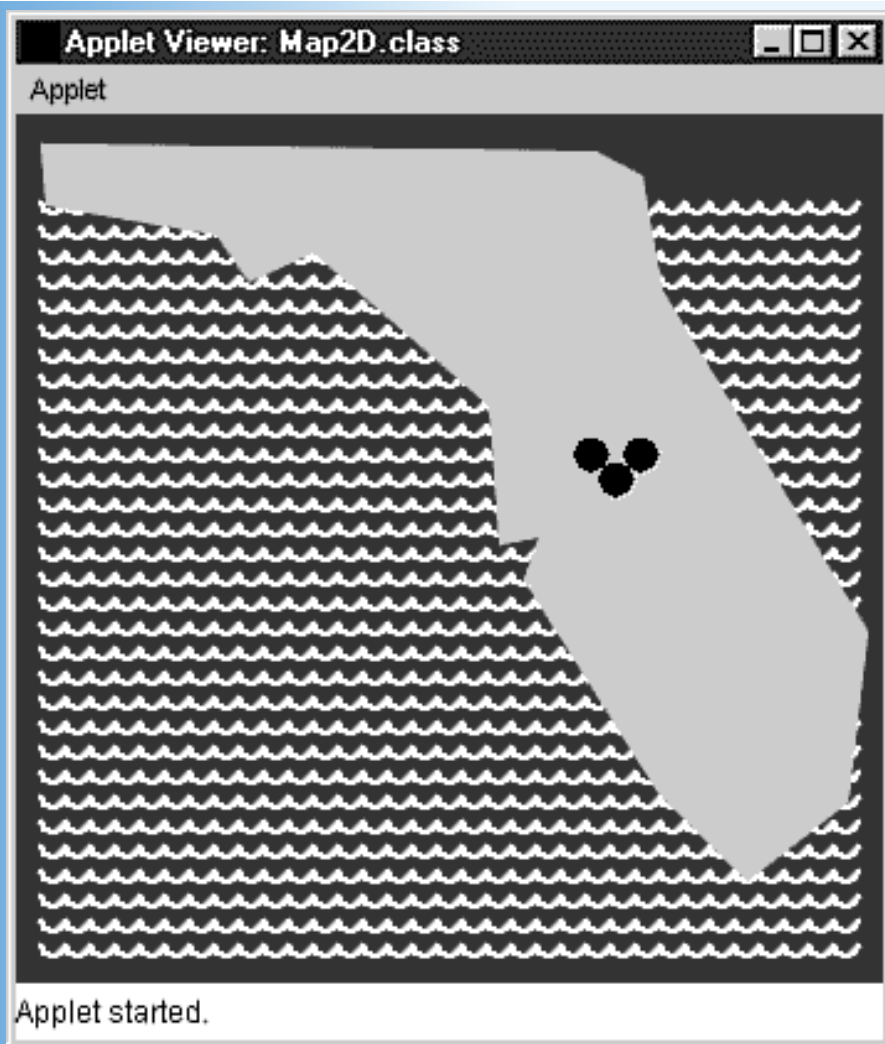


Abbildung 9.17:
Das Map2D-Applet

Zusammenfassung

Sie verfügen nun über einige Tools, mit denen Sie das Erscheinungsbild eines Applets verbessern können. Sie können mit Linien, Rechtecken, Ellipsen, Polygonen, Schriften, Farben und Mustern auf einem Applet-Fenster arbeiten, indem Sie Nicht-2D- und 2D-Klassen verwenden.

Bei Nicht-2D-Zeichenoperationen verwenden Sie Methoden der Klasse Graphics mit Argumenten, die das zu zeichnende Objekt beschreiben.

Java2D verwendet für jede Zeichenoperation dieselben beiden Methoden - draw() und fill(). Unterschiedliche Objekte werden mit den Klassen aus dem Paket java.awt.geom erzeugt. Diese werden dann als Argumente für die Zeichenmethoden von Graphics2D verwendet.

Später in diesem Buch lernen Sie, wie Sie auf andere Komponenten eines Java-Programms in der Art, wie Sie es mit dem Applet-Fenster getan haben, zeichnen. Dies ermöglicht es Ihnen, die Techniken des heutigen Tages auch in Java-Applikationen zu verwenden.

Sie werden morgen noch mehr Gelegenheit erhalten, Onkel Walter zu beeindrucken, wenn der Kunstunterricht Animation und die Anzeige von Bilddateien behandelt.

Fragen und Antworten

Frage:

Ich will eine Textzeile ausgeben, in deren Mitte sich ein fett gedrucktes Wort befindet. Ich verstehe, daß ich zwei verschiedene Font-Objekte dafür brauche - eines für die normale Schrift und eines für die fette - und daß ich die aktuelle Schrift mittendrin umdefinieren muß. Das Problem ist nun, daß drawString() eine x- und eine y-Position für den Beginn eines jeden Strings benötigt, und ich kann nichts finden, was sich auf die aktuelle Position innerhalb des Applet-Fensters bezieht. Wie kann ich ermitteln, wo das fett gedruckte Wort starten soll?.

Antwort:

Die Darstellungsmöglichkeiten von Java für Text sind ziemlich primitiv. Java hat nichts in der Art eines aktuellen Punktes, so daß Sie selbst ermitteln müssen, wo sich das Ende des einen Strings befindet, um den nächsten String zu beginnen. Die Methode `stringWidth()` kann Ihnen bei diesem Problem helfen - sowohl, um die Länge des gerade ausgegebenen Strings zu ermitteln, als auch, um Leerraum danach einzufügen.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).

Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 2

Tag 10

Bilder, Sound und Animation

Den ersten Kontakt mit Java stellte für die meisten Leute animierter Text oder bewegte Bilder auf einer Webseite dar. Diese Animationsarten sind einfach, da sie hierfür nur wenige Methoden in Java implementieren müssen. Diese Methoden sind allerdings die Basis für jedes Applet, das den Bildschirminhalt dynamisch aktualisiert. Mit einfacher Animation zu beginnen, stellt einen guten Weg dar, komplexere Applets zu erstellen.

Animationen werden unter Java mit bestimmten Klassen und Methoden des *Abstract Windowing Toolkit* (AWT) umgesetzt. Heute lernen Sie, wie die verschiedenen Teile von Java zusammenarbeiten, so daß Sie bewegte Bilder erstellen und Applets dynamisch aktualisieren können.

Animationen zu erzeugen macht Spaß und ist in Java einfach. Es läßt sich schon mit den integrierten Methoden von Java für Linien, Schriften und Farben sehr viel machen. Für interessante Animationen benötigen Sie ein eigenes Bild für jedes Einzelbild der Animation - Sound dazu zu haben ist auch nett.

Heute lernen Sie die folgenden Themen kennen:

- Wie Animationen unter Java funktionieren - die Methoden `paint()` und `repaint()`, dynamische Applets starten und stoppen und wie Sie diese Methoden in Ihren eigenen Applets überschreiben.
- Threads - was Threads sind und wie sie Ihrem Applet zu einem wohlherzogenen Umgang mit anderen Applets und dem System im allgemeinen verhelfen.
- Methoden zur Reduzierung von Flimmereffekten - ein weitverbreitetes Problem bei Animationen in Java.
- Bitmap-Bilder wie z.B. GIFs oder JPEGs verwenden - diese vom Server abrufen, unter Java laden, in Ihrem Applet anzeigen und sie in Animationen verwenden.
- Sounds anwenden - Herunterladen und Abspielen zu geeigneten Zeitpunkten.

Animationen unter Java erstellen

Animationen sind unter Java ein verhältnismäßig einfacher Prozeß, der die folgenden Schritte umfaßt:

- Etwas mit Text, Objekten oder Bildern zeichnen.
- Das Ausgabesystem anweisen, daß es Ihre »Zeichnungen« ausgeben soll.

Dieses Schritte werden mit unterschiedlichen Dingen, die gezeichnet werden sollen, wiederholt, so daß der Eindruck der Bewegung entsteht. Sie können das Zeitintervall zwischen den Einzel./Bildern der Animation verändern oder Java anweisen, diese so schnell wie möglich auszugeben.

Zeichnen und Neuzeichnen

Wie Sie bereits gelernt haben, wird die `paint()`-Methode aufgerufen, wenn der Ausgabebereich eines Applets neu gezeichnet werden muß. Diese Methode wird aufgerufen, wenn ein Applet startet, da das Applet-Fenster leer ist und der Inhalt das erste Mal dargestellt werden muß. Sie wird auch aufgerufen, wenn das Applet-Fenster in den Vordergrund kommt, nachdem es von dem Fenster eines anderen Programms verdeckt wurde.

Sie können das Ausgabesystem von Java bitten, das Fenster neu zu zeichnen, indem Sie die Methode `repaint()`

aufrufen.



Diese höfliche Ausdrucksform wurde hier aus einem ganz bestimmten Grund gewählt - `repaint()` stellt nämlich wirklich eher eine Anfrage als ein Kommando dar. Das Ausgabesystem von Java erhält diese Anfrage und verarbeitet sie, sobald dies möglich ist. Sollten die `repaint()`-Anfragen schneller auflaufen, als Java diese verarbeiten kann, werden eventuell einige übersprungen. In den meisten Fällen ist die Verzögerung zwischen dem Aufruf von `repaint()` und der eigentlichen Aktualisierung des Fensters vernachlässigbar.

Um das Erscheinungsbild dessen, was in einem Applet-Fenster angezeigt wird, zu ändern, zeichnen Sie die gewünschten Dinge und rufen `repaint()` auf. Anschließend zeichnen Sie etwas anderes und rufen `repaint()` erneut auf usw.

All dies findet nicht in der `paint()`-Methode statt, da diese nur für die Ausgabe eines Einzelbildes verantwortlich ist - nämlich das aktuelle. Die eigentliche Arbeit wird an anderer Stelle im Applet verrichtet.

An dieser anderen Stelle, die durchaus eine eigene Methode sein könnte, erstellen Sie Objekte, zeichnen diese und verrichten andere notwendige Aufgaben. Abschließend rufen Sie die Methode `repaint()` auf.



Obwohl Sie die Methode `paint()` auch selbst aufrufen können, sollten Sie für alle Anfragen zum Zeichnen des Ausgabebereiches Aufrufe der Methode `repaint()` verwenden. Die `repaint()`-Methode ist leichter anzuwenden - sie benötigt kein `Graphics`-Objekt als Parameter, wie das bei `paint()` der Fall ist - und Sie kümmert sich um alles, was zur Aktualisierung des Anzeigebereiches nötig ist. Sie werden dies später am heutigen Tag sehen, wenn Sie `repaint()` verwenden, um eine Animationssequenz zu erstellen.

Ein Applet starten und stoppen

Wie Sie sich von Tag 8 her erinnern werden, wird die `start()`-Methode beim Start eines Applets und die `stop()`-Methode beim Beenden eines Applets aufgerufen.

Diese Methoden sind leer, wenn Sie sie von der Klasse `java.applet.Applet` erben, so daß Sie diese überschreiben müssen, damit sie beim Start bzw. beim Beenden Ihres Programms etwas tun. Sie haben `start()` und `stop()` gestern nicht verwendet, da Sie die Applets `paint()` nur einmal verwendet haben.

Bei Animationen und anderen Java-Applets, die längere Zeit laufen, werden `start()` und `stop()` benötigt, um den Start Ihres Applets auszulösen und die Ausführung wieder zu beenden, sobald die Seite, auf der sich das Applet befindet, verlassen wird.

Animationen über Threads kontrollieren

Animationen stellen eine ideale Anwendung für Threads dar, Javas Möglichkeit für die Verarbeitung von mehr als einer Aufgabe zur selben Zeit.



Ein *Thread* ist ein Teil eines Programms, der eingerichtet wird, um eigenständig zu laufen, während der Rest des Programms etwas anderes tut. Dies wird auch als *Multitasking* bezeichnet, da das Programm mehr als eine

Aufgabe zur selben Zeit ausführen kann.

Threads sind ideal für alles, was viel Rechenzeit in Anspruch nimmt und kontinuierlich ausgeführt wird, wie z.B. die wiederholten Zeichenoperationen, die eine Animation ausmachen.

Indem Sie die Arbeitslast der Animation in einen Thread packen, machen Sie den Weg dafür frei, daß sich der Rest des Programms mit anderen Dingen beschäftigen kann. Sie machen es auch für die Laufzeitumgebung des Applets einfacher, das Programm zu verarbeiten, da die gesamte Rechen- und zeitintensive Arbeit in ihrem eigenen Thread isoliert ist.

Applets mit Threads schreiben

Um einen Thread in einem Applet zu verwenden, können Sie fünf Veränderungen an dessen Quellcode vornehmen:

- Fügen Sie in die Deklaration der Klasse die Anweisung `implements Runnable` ein.
- Erzeugen Sie ein Thread-Objekt, das den Thread aufnimmt.
- Überschreiben Sie die `start()`-Methode, um den laufenden Thread auf null zu setzen.
- Erstellen Sie eine `run()`-Methode, die alle Anweisungen enthält, um das Applet kontinuierlich laufen zu lassen.

Das Schlüsselwort `implements` ähnelt dem Schlüsselwort `extends`, da es die Klasse verändert, die in derselben Zeile deklariert wird. Im folgenden sehen Sie ein Beispiel für eine Klasse, die sowohl `extends` als auch `implements` verwendet:

```
public class DancingBaby extends java.applet.Applet
    implements Runnable {
    // ...
}
```

Obwohl die Deklaration der Klasse in zwei Zeilen aufgeteilt wurde, deklariert alles vom Schlüsselwort `public` bis hin zur geschweiften Klammer `»{«` die Klasse.

`Runnable` ist eine besondere Art von Klasse, die als Schnittstelle bezeichnet wird. Wie Sie sich von vielleicht von Tag 2 her erinnern werden, stellen Schnittstellen für Klassen einen Weg dar, Methoden zu erben, die sie ansonsten nicht von deren Superklasse erben könnten.

Diese Methoden können von jeder beliebigen Klasse implementiert werden, die diese Verhaltensweisen benötigt. In diesem Beispiel wird die `Runnable`-Schnittstelle von einer Klasse implementiert, die als Thread arbeiten soll. `Runnable` bietet eine Deklaration für die Methode `run()`, die für den Start eines Threads aufgerufen wird.

Die `Thread`-Klasse ist Bestandteil des Paketes `java.lang`, so daß Sie diese nicht über eine `import`-Anweisung verfügbar machen müssen. Der Anfang bei der Erstellung eines Threads ist sehr einfach - lediglich die Vergabe eines Namens ist erforderlich. Dies ist im folgenden Beispiel gezeigt:

```
Thread runner;
```

Dieses Objekt kann in der `start()`-Methode des Applets erzeugt werden. Die Variable `runner` hat den Wert null, bis das Objekt erzeugt wird.

Der ideale Ort, es zu erzeugen, ist die `start()`-Methode des Applets. Die folgende Methode prüft, ob der Thread bereits erzeugt wurde. Ist dies nicht der Fall, erzeugt sie diesen:

```
public void start() {
    if (runner == null) {
        runner = new Thread(this);
        runner.start();
    }
}
```

```
    }
}
```

Das Schlüsselwort `this`, das im Konstruktor `Thread()` verwendet wird, ist eine Möglichkeit, sich auf das Objekt, das die Methode ausführt - das Applet selbst in diesem Fall - zu beziehen. Indem Sie `this` verwenden, wird das Applet als die Klasse identifiziert, die die benötigten Verhaltensweisen für die Ausführung des Threads enthält.

Um einen Thread auszuführen, rufen Sie dessen `start()`-Methode auf, wie das in der folgenden Anweisung aus vorherigem Beispiel der Fall ist:

```
runner.start();
```

Der Aufruf der `start()`-Methode des Threads hat zur Folge, daß eine weitere Methode aufgerufen wird - die `run()`-Methode der Klasse, die den Thread beinhaltet.

In diesem Beispiel implementiert das Applet die `Runnable`-Schnittstelle und wurde mit dem `runner`-Objekt über das Schlüsselwort `this` verknüpft. Eine Methode mit dem Namen `run()` muß in das Applet eingefügt werden. Im folgenden ein Beispiel:

```
public void run() {
    // was Ihr Applet eigentlich tut
}
```

Die `run()`-Methode ist das Herz eines Applets, das mit Threads arbeitet. Sie sollte dazu verwendet werden, eine Animationssequenz zu steuern. Hier sollte alles vorgenommen werden, was für die Zeichnungen und zum Ändern der Dinge zwischen den Einzel./Bildern nötig ist.

Nachdem die `run()`-Methode mit allen Verhaltensweisen, die der Thread benötigt, ausgestattet wurde, ist der letzte Schritt dabei, das Applet threadfähig zu machen und den Thread über seine `stop()`-Methode zu beenden.

Einen Thread stoppen Sie, indem Sie sein Objekt auf `null` setzen. Dies beendet den Thread nicht. Allerdings können Sie die `run()`-Methode so gestalten, daß sie nur läuft, solange das Thread-Objekt verschieden von `null` ist.



Es gibt eine `stop()`-Methode, die zum Beenden von Threads aufgerufen werden kann. JavaSoft hat diese allerdings mit der Version 1.2 von Java verworfen. Durch die Verwendung der `stop()`-Methode des Threads entstehen Instabilitäten in der Laufzeitumgebung des Programms. Außerdem kann es bei dessen Ausführung zu Fehlern kommen, die nur schwer aufzudecken sind. Es wird den Programmierern stark davon abgeraten, mit `stop()` einen Thread unter Java zu stoppen, sogar in Java 1.02- und Java- 1.1-Programmen.

Indem Sie implements `Runnable` hinzufügen, ein Thread-Objekt erzeugen, das dem Applet zugeordnet ist, und die Methoden `start()`, `stop()` und `run()` des Applets verwenden, wird ein Applet zu einem Programm, das Threads verwendet.

Die Teile zusammenfügen

Die Programmierung mit Threads sollte klarer werden, wenn Sie dies direkt in Aktion sehen. Listing 10.1 beinhaltet ein einfaches animiertes Applet, das das Datum und die Zeit anzeigt. Die Darstellung wird in konstanten Intervallen aktualisiert. Dadurch ergibt sich eine Digitaluhr (siehe Abbildung 10.1).

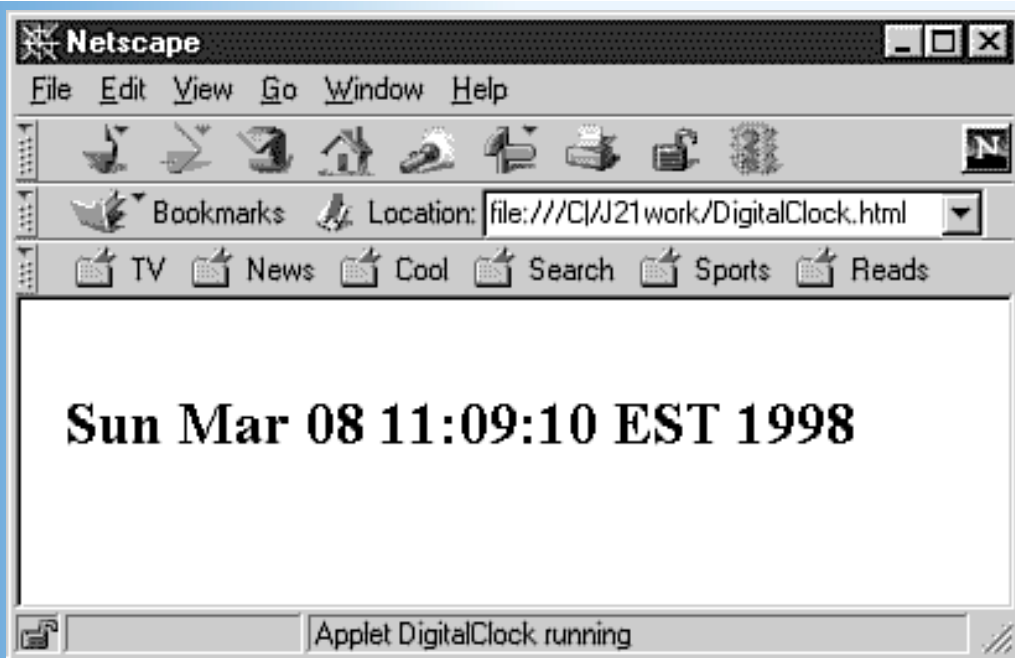


Abbildung 10.1:
Das DigitalClock-Applet im Netscape Navigator

Dieses Applet verwendet die Methoden `paint()`, `start()` und `stop()`. Außerdem verwendet es Threads.

Listing 10.1: Der gesamte Quelltext von DigitalClock.java

```

1: import java.awt.Graphics;
2: import java.awt.Font;
3: import java.util.Date;
4:
5: public class DigitalClock extends java.applet.Applet
6:     implements Runnable {
7:
8:     Font theFont = new Font("TimesRoman",Font.BOLD,24);
9:     Date theDate;
10:    Thread runner;
11:
12:    public void start() {
13:        if (runner == null) {
14:            runner = new Thread(this);
15:            runner.start();
16:        }
17:    }
18:
19:    public void stop() {
20:        if (runner != null) {
21:            runner = null;
22:        }
23:    }
24:
25:    public void run() {
26:        Thread thisThread = Thread.currentThread();
27:        while (runner == thisThread) {
28:            repaint();
29:            try {
30:                Thread.sleep(1000);

```

```

31:         } catch (InterruptedException e) { }
32:     }
33: }
34:
35: public void paint(Graphics screen) {
36:     theDate = new Date();
37:     screen.setFont(theFont);
38:     screen.drawString("" + theDate.toString(), 10, 50);
39: }
40: }

```

Um das Applet zu testen, fügen Sie es auf einer Webseite in einem Applet-Fenster mit den folgenden Attributen ein: width=380 und height=100.



Dieses Applet verwendet die Date-Klasse, um das aktuelle Datum und die Uhrzeit zu ermitteln. Dadurch wird das Applet kompatibel zu Java 1.02. In den neueren Versionen der Sprache sollten die Klassen Calendar und GregorianCalendar verwendet werden, da diese eine bessere Unterstützung internationaler Kalendersysteme bieten. Auf der CD zum Buch finden Sie eine Java-1.2-konforme Version des DigitalClock-Applets (DigitalClock12.java)

Animationen sind ein gutes Beispiel für die Art der Aufgaben, die einen eigenen Thread benötigen. Sehen Sie sich einmal die endlose while-Schleife im DigitalClock- Applet an. Wenn Sie nicht mit Threads arbeiten würden, würde die while-Schleife im Standard-Java-System-Thread laufen, der auch für die Ausgabe auf dem Bildschirm, die Verarbeitung von Benutzereingaben, wie z.B. Mausklicks, und dafür, daß intern alles aktuell ist, verantwortlich ist. Unglücklicherweise reißt die while-Schleife alle Java- Ressourcen an sich, wenn sie im Hauptsystem-Thread ausgeführt wird, und hält alles andere - inklusive der Bildschirmausgabe - davon ab, ausgeführt zu werden. Sie würden nichts auf dem Bildschirm sehen, da Java abwarten würde, bis die while-Schleife verarbeitet ist, bevor es irgend etwas anderes tut.

Sie betrachten in diesem Abschnitt das Applet aus der Perspektive der Teile der eigentlichen Animation. Anschließend werden Sie sich mit den Teilen beschäftigen, die die Threads verwalten.

Die Zeilen 8-9 definieren zwei Instanzvariablen: theFont und theDate. Diese nehmen Objekte auf, die die aktuelle Schrift bzw. das aktuelle Datum repräsentieren. Darüber erfahren Sie später mehr.

Die Methoden start() und stop() starten bzw. stoppen den Thread. Der wesentliche Teil der Arbeit wird in der run()-Methode (Zeilen 25-33) ausgeführt.

Innerhalb der run()-Methode findet die eigentliche Animation statt. Sehen Sie sich die while-Schleife in dieser Methode an (beginnend mit der Anweisung in Zeile 27). Der Ausdruck runner == thisThread gibt den Wert true zurück, bis das Objekt runner auf null gesetzt wird (dies geschieht in der stop()-Methode des Applets). In der Schleife wird ein Einzelbild der Animation erstellt.

Als erstes wird in der Schleife die Methode repaint() aufgerufen (Zeile 28), um das Applet neu auszugeben. In den Zeilen 29-31, so kompliziert diese auch erscheinen mögen, passiert nichts anderes, als daß vor der nächsten Schleifenwiederholung eine Pause von 1000 Millisekunden (1 Sekunde) eingelegt wird.

Die sleep()-Methode der Klasse Thread sorgt dafür, daß das Applet pausiert. Ohne die sleep()-Methode würde das Applet so schnell wie möglich ausgeführt werden. Die sleep()-Methode kontrolliert genau, wie schnell die Animation abläuft. Die try- und catch-Sachen rundherum ermöglichen es Java, Fehler, falls welche auftreten, zu behandeln. Diese Anweisungen werden an Tag 17 beschrieben.

In der `paint()`-Methode in den Zeilen 35-39 wird eine neue Instanz der Klasse `Date` erzeugt. Diese enthält das aktuelle Datum und die aktuelle Uhrzeit - beachten Sie bitte, daß diese Klasse in der Zeile 3 explizit importiert wurde. Dieses neue `Date`-Objekt wird der Instanzvariablen `theDate` zugewiesen.

In der Zeile 37 wird die aktuelle Schrift gesetzt. Dazu wird der Wert der Variablen `theFont` verwendet. Außerdem wird das Datum auf dem Bildschirm ausgegeben - beachten Sie bitte, daß Sie die Methode `toString()` der Klasse `Date` aufrufen müssen, um das Datum und die Zeit als String anzeigen zu lassen. Jedesmal, wenn `paint()` aufgerufen wird, wird ein neues `theDate`-Objekt erzeugt, das das aktuelle Datum und die Uhrzeit enthält.

Betrachten Sie nun die Code-Zeilen dieses Applets, die den Thread erzeugen und verwalten. Werfen Sie als erstes einen Blick auf die Deklaration der Klasse selbst in den Zeilen 5-6. Beachten Sie, daß die Klassendeklaration die Schnittstelle `Runnable` implementiert. Jede Klasse, die Sie erstellen und die Threads verwendet, muß `Runnable` beinhalten.

Zeile 10 definiert eine dritte Instanzvariable für diese Klasse, die `runner` genannt wird und den Typ `Thread` hat. Diese nimmt das `Thread`-Objekt für dieses Applet auf.

In den Zeilen 12-23 werden die geerbten Methoden `start()` und `stop()` definiert. Diese tun nichts, außer Threads zu erzeugen und zu zerstören. Diese Methodendefinitionen werden bei den meisten Klassen ähnlich sein, da sie lediglich die Infrastruktur die Threads, die von einem Programm verwendet werden, einrichten.

Zum Schluß noch die `run()`-Methode, in der die meiste Arbeit in Ihrem Applet verrichtet wird (Zeilen 25-33).

Das Flimmern in Animationen reduzieren

Wenn das `DigitalClock`-Applet ausgeführt wird, sehen Sie gelegentlich ein Flimmern in dem Text, den es anzeigt. Das Ausmaß des Flimmerns hängt von der Qualität der Java-Laufzeitumgebung ab, in der das Programm ausgeführt wird, wie auch von der Prozessorgeschwindigkeit. Allerdings ist es wahrscheinlich selbst auf schnellen PCs mit gut implementierter Java Virtual Machine störend.

Flimmern ist einer der Nebeneffekte der Art, wie der Ausgabebereich in einem Java- Programm aktualisiert wird. Und es ist eines der Probleme, auf die Sie bei der Erzeugung einer Animation stoßen werden.

Flimmern und wie Sie es vermeiden

Flimmern wird von der Art, wie Java jedes Einzelbild einer Animation darstellt, verursacht. Zu Beginn der heutigen Lektion haben Sie gelernt, daß bei einem Aufruf der `repaint()`-Methode diese die Methode `paint()` aufruft.

Tatsächlich ist hier aber noch ein Mittelsmann beteiligt. Wenn `repaint()` aufgerufen wird, ruft sie die Methode `update()` auf, die das Applet-Fenster von allen vorhandenen Inhalten befreit, indem Sie es mit dessen aktueller Hintergrundfarbe füllt. Die `update()` -Methode ruft anschließend die `paint()`-Methode auf.

Das Löschen des Bildschirminhalts in der Methode `update()` ist der Übeltäter in bezug auf das Flimmerproblem. Da das Applet-Fenster zwischen den Einzel./Bildern gelöscht wird, springen die Bereiche des Applet-Fensters, die sich nicht ändern, kurz zwischen dem Zustand *gelöscht* und *neugezeichnet* hin und her - mit anderen Worten sie flimmern.

Es gibt zwei Hauptmethoden, um das Flimmern in Ihren Java-Applets zu vermeiden:

- Überschreiben Sie die `update()`-Methode, so daß sie entweder den Bildschirm überhaupt nicht löscht oder nur die Teile löscht, die Sie geändert haben.
- Überschreiben Sie sowohl die Methoden `update()` und `paint()`, und verwenden Sie die doppelte Pufferung.

Der einfachste Weg das Flimmern zu reduzieren, ist, die `update()`-Methode so zu überschreiben, daß sie den Bildschirm nicht löscht. Der erfolgreichste Weg, sich des Problems anzunehmen, ist allerdings die doppelte Pufferung.

So überschreiben Sie `update()`

Die standard `update()`-Methode jedes Applets hat die folgende Form:

```
public void update(Graphics g) {
    g.setColor(getBackground());
    g.fillRect(0, 0, size().width, size().height);
    g.setColor(getForeground());
    paint(g);
}
```

Die `update()`-Methode löscht den Bildschirm, indem sie das Applet-Fenster mit der Hintergrundfarbe füllt, die aktuelle Farbe auf die Vordergrundfarbe setzt und anschließend `paint()` aufruft. Wenn Sie `update()` mit Ihrer eigenen Version überschreiben, müssen Sie sicherstellen, daß Ihre Version etwas Ähnliches macht. In den beiden folgenden Abschnitten arbeiten Sie ein paar Beispiele durch, in denen Sie `update()` überschreiben, um das Flimmern zu reduzieren.

Lösung eins: Löschen Sie den Bildschirm nicht

Die erste Lösung, um das Flimmern zu reduzieren, ist, den Bildschirm überhaupt nicht zu löschen. Diese Lösung funktioniert natürlich nur bei wenigen Applets. Das `ColorSwirl`-Applet z.B. zeigt einen einzigen String (Look to the cookie!) an. Dieser String wird allerdings in unterschiedlichen Farben angezeigt, die dynamisch in andere übergehen. Dieses Applet flimmert fächerlich, wenn man es ausführt. In Listing 10.2 sehen Sie den ursprünglichen Quellcode für dieses Applet, und Abbildung 10.2 zeigt das Ergebnis.

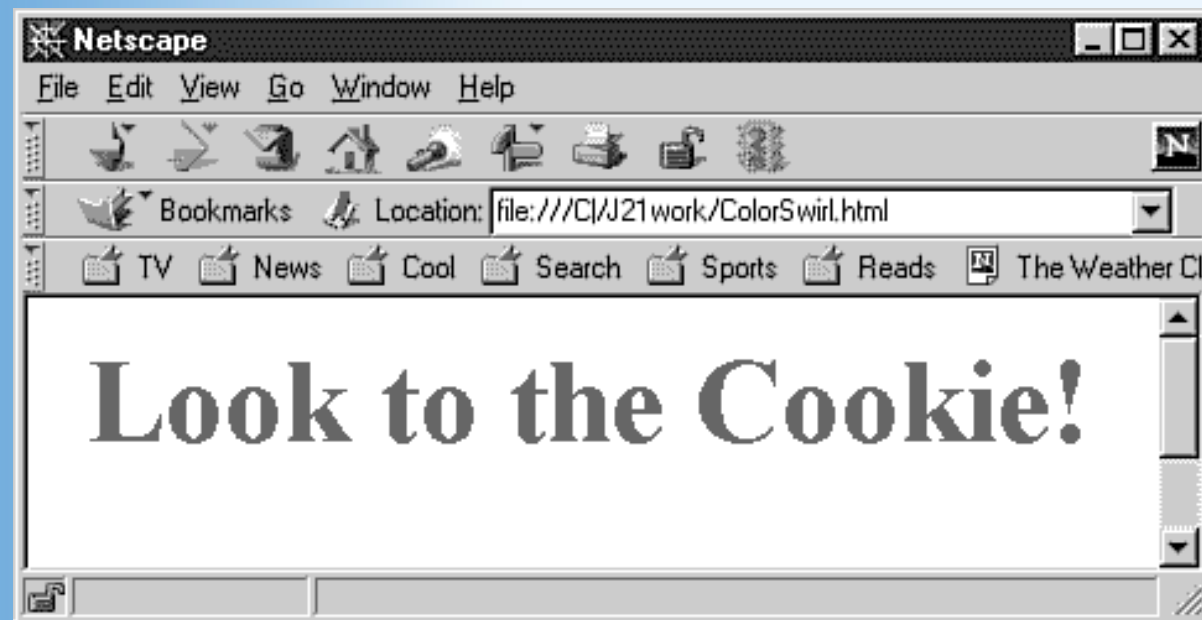


Abbildung 10.2:
Die Ausgabe des `ColorSwirl`-Applets im Netscape Navigator

Listing 10.2: Der gesamte Quelltext von `ColorSwirl.java`

```
1: import java.awt.Graphics;
2: import java.awt.Color;
3: import java.awt.Font;
4:
```

```

5: public class ColorSwirl extends java.applet.Applet
6:     implements Runnable {
7:
8:     Font f = new Font("TimesRoman", Font.BOLD, 48);
9:     Color colors[] = new Color[50];
10:    Thread runner;
11:
12:    public void start() {
13:        if (runner == null) {
14:            runner = new Thread(this);
15:            runner.start();
16:        }
17:    }
18:
19:    public void stop() {
20:        runner = null;
21:    }
22:
23:    public void run() {
24:        // Das Array der Farben initialisieren
25:        float c = 0;
26:        for (int i = 0; i < colors.length; i++) {
27:            colors[i] =
28:                Color.getHSBColor(c, (float)1.0,(float)1.0);
29:            c += .02;
30:        }
31:
32:        // Die einzelnen Farben durchgehen
33:        int i = 0;
34:        Thread thisThread = Thread.currentThread();
35:        while (runner == thisThread) {
36:            setForeground(colors[i]);
37:            repaint();
38:
39:            i++;
40:            try {
41:                Thread.sleep(200);
42:            } catch (InterruptedException e) { }
43:            if (i == colors.length ) i = 0;
44:        }
45:    }
46:
47:    public void paint(Graphics screen) {
48:        screen.setFont(f);
49:        screen.drawString("Look to the Cookie!", 15, 50);
50:    }
51: }

```

Um dieses Applet zu testen, fügen Sie es in eine Webseite ein mit den folgenden Größen-Attributen im `<APPLET>`-Tag: height=150 width=450. Drei Dinge werden Ihnen bei diesem Applet vielleicht seltsam erscheinen:

- Zeile 9 definiert eine Instanzvariable namens colors. Diese stellt ein Array mit 50 Elementen dar. Wenn das Applet mit der Ausführung beginnt, wird in der run()- Methode (in den Zeilen 25-30) als erstes

dieses Array mit Color-Objekten gefüllt. Indem Sie alle benötigten Farben zu Anfang erzeugen, können Sie Text in diesen Farben - eine nach der anderen - ausgeben. Es ist einfacher, all diese Farben auf einmal zu berechnen. (Eigentlich würde diese for-Schleife in der init()-Methode mehr Sinn ergeben, da sie nur einmal ausgeführt wird. Die Entscheidung, 50 Farben zu verwenden, ist willkürlich - das Programm könnte genauso einfach mit 20 oder mit 250 Farben arbeiten.

- Um die unterschiedlichen Color-Objekte zu erzeugen, wird eine Methode der Klasse Color - getHSBColor() - verwendet, statt new mit diversen sRGB-Werten aufzurufen. Die Klassenmethode getHSBColor() erzeugt ein Color-Objekt, basierend auf den Werten für Farbton, Sättigung und Helligkeit. Indem man den Farbton inkrementiert, während man die Werte für Sättigung und Helligkeit konstant läßt, kann man einen Bereich von Farben erzeugen, ohne für jede einzelne die entsprechenden sRGB-Werte erzeugen, zu müssen. Es ist lediglich ein schneller und einfacher Weg, das colors-Array zu erzeugen.
- Um eine Animation zu erzeugen, geht das Applet das Array mit den Farb-Objekten durch, setzt die Vordergrundfarbe auf die Farbe, die an der Reihe ist, und ruft repaint() auf. Wenn das Ende des Arrays erreicht ist, fängt das Applet wieder am Anfang des Arrays an (siehe Zeile 45), so daß sich der Prozeß wiederholt - bis in alle Ewigkeit.

Da Sie nun verstehen, was das Applet tut, ist es an der Zeit, das Flimmerproblem zu beheben. Das Flimmern entsteht, da es bei jeder Ausgabe des Applets einen Moment gibt, in dem der Bildschirm gelöscht wird. Anstatt daß der Text nahtlos von Rot zu Rosa nach Violett übergeht, geht er von Rot nach Grau nach Pink nach Grau nach Violett nach Grau usw. über.

Da lediglich das Löschen des Bildschirms dieses Problem verursacht, ist die Lösung einfach: Überschreiben Sie update(), und entfernen Sie den Teil, in dem der Bildschirm gelöscht wird. Es besteht kein Bedarf für das Löschen, da sich hier nichts außer der Farbe des Textes ändert. Dadurch, daß Sie das Löschen des Bildschirms aus update() entfernen, muß update() nur noch die paint()-Methode aufrufen. Im folgenden die update()-Methode, wie Sie in dem überarbeiteten ColorSwirl-Applet aussehen sollte:

```
public void update(Graphics screen) {
    paint(screen);
}
```

Indem Sie diese drei Zeilen hinzufügen, beenden Sie das Flimmern des Applets.



Sie finden die erste Version von ColorSwirl.java unter diesem Namen auf der CD des Buches und die verbesserte Version an derselben Stelle mit dem Namen BetterSwirl.java .

Sie lernen heute noch eine andere Methode zur Reduzierung des Flimmerns kennen - eine Technik, die als doppelte Pufferung bezeichnet wird.

Bilder laden und anzeigen

Der elementare Umgang mit Bildern wird unter Java von der Klasse Image geboten, die Teil des Paketes java.awt ist. Wenn Sie mit einem Applet arbeiten, können Sie Methoden der Klassen Applet und Graphics verwenden, um Bilder zu laden und anzuzeigen.

Bilder laden

Um ein Bild in Ihrem Applet anzeigen zu können, müssen Sie dies erst über das World Wide Web in Ihr Java-Programm laden. Bilder werden als separate Dateien außerhalb der .class-Dateien von Java gespeichert.

Aus diesem Grund müssen Sie Java mitteilen, wo es diese Dateien findet.

Wenn Sie die Image-Klasse verwenden, muß das Bild im Format .GIF oder .JPG vorliegen.

Eine Adresse im Web wird unter Java von einem URL-Objekt repräsentiert. Das Akronym URL steht für *Uniform Resource Locator*. Die Klasse URL ist Teil des Paketes java.net, so daß Sie für diese Klasse wie schon bei der Image-Klasse eine import-Anweisung in Ihr Programm einfügen müssen.

Das URL-Objekt wird erzeugt, indem eine Web-Adresse an den Konstruktor der Klasse URL übergeben wird. Im folgenden ein Beispiel:

```
URL u = new URL("http://www.prefect.com/java21/images/book.gif");
```

Wenn Sie ein URL-Objekt haben, können Sie es dazu verwenden, ein Image-Objekt zu erzeugen, das die Grafikdatei repräsentiert.

Die Applet-Klasse bietet eine Methode namens getImage(), mit der ein Bild in ein Image-Objekt geladen werden kann. Es gibt zwei Möglichkeiten, diese Methode zu verwenden:

- Die Methode getImage(), aufgerufen mit einem einzigen Argument (ein Objekt vom Typ URL), lädt das Bild bei dieser URL.
- Die Methode getImage(), aufgerufen mit zwei Argumenten (der Basis-URL des Bildes - auch ein URL-Objekt - und einem String, der den relativen Pfad oder Dateinamen des aktuellen Bildes angibt).

Obwohl der erste Weg einfacher erscheint, ist der zweite der flexiblere. Wenn Sie eine bestimmte Web-Adresse in Ihrem Applet verwenden, müssen Sie das Programm verändern und neu kompilieren, sobald Ihre Web-Site umzieht.

Die Klasse Applet hat zwei Methoden, mit denen man eine Basis-URL erzeugen kann, ohne eine feste Adresse im Programm angeben zu müssen:

- Die Methode getDocumentBase() gibt ein URL-Objekt zurück, das den Ordner repräsentiert, der die Webseite mit dem Applet enthält. Wenn die Seite z.B. unter http://www.prefect.com/java21/ zu finden ist, gibt getDocumentBase() die URL zurück, die auf diesen Pfad verweist.
- Die Methode getCodeBase() gibt ein URL-Objekt zurück, das den Ordner repräsentiert, in dem sich die .class-Datei der Hauptklasse des Applets befindet.

Relative Pfadangaben

Der relative Pfad, den Sie als zweites Argument in getImage() verwenden, hängt davon ab, was Sie als erstes Argument verwendet haben.

Nehmen Sie z.B. eine Webseite unter der Adresse http://www.prefect.com/java21/ index.html, die eine Bilddatei mit der URL http://www.prefect.com/java21/ book.gif hat. Um dieses Bild in ein Applet zu laden, könnten Sie die folgende Anweisung verwenden:

```
Image img = new URL(getDocumentBase(), "book.gif");
```

Wenn, als weiteres Beispiel, die Bilddatei an die Adresse http://www.prefect.com/ java21/images/book.gif verschoben werden würde, könnten Sie die folgende Anweisung verwenden:

```
Image img = new URL(getDocumentBase(), "images/book.gif");
```

Ob Sie getDocumentBase() oder getCodeBase() verwenden, hängt davon ab, ob Sie Ihre Bilder in Unterordnern Ihres Java-Applets oder in Unterordnern der Webseite des Applets speichern.



Wenn Sie `getDocumentBase()` oder `getCodeBase()` verwenden, können Sie die Bilder auch laden, wenn Sie das Applet auf Ihrem eigenen Computer testen. Sie müssen es nicht auf einer Site im World Wide Web speichern, um feststellen zu können, ob es funktioniert.

Indem Sie eine dieser Methoden verwenden, machen Sie es möglich, daß das Applet mit seiner Webseite umzieht und Sie keine Änderungen am Programm vornehmen müssen.



Wenn Sie ein Java-Archiv (eine .JAR-Datei) verwenden, um Ihr Applet zum Benutzer zu bringen, können Sie Bilddateien und andere Datendateien in dem Archiv ablegen. Diese Dateien werden aus dem Archiv mit den .class-Dateien in den .JAR-Dateien automatisch extrahiert.

Bilder ausgeben

Nachdem Sie ein Bild in ein Image-Objekt geladen haben, können Sie es in einem Applet mit der Methode `drawImage()` der `Graphics`-Klasse anzeigen.

Um ein Bild mit seiner Originalgröße anzuzeigen, rufen Sie die `drawImage()`-Methode mit vier Argumenten auf:

- Das Image-Objekt, das angezeigt werden soll
- Die x-Koordinate
- Die y-Koordinate
- Das Schlüsselwort `this`

Ist eine Grafikdatei in dem `img`-Objekt gespeichert, kann die folgende `paint()`-Methode zur Anzeige verwendet werden:

```
public void paint(Graphics screen) {
    screen.drawImage(img, 10, 10, this);
}
```

Die x,y-Koordinaten, die der `drawImage()`-Methode übergeben werden, sind mit den x,y-Koordinaten vergleichbar, die bei der Anzeige eines Rechteckes verwendet werden. Der Punkt repräsentiert die linke, obere Ecke des Bildes.

Sie können ein Bild in einer anderen Größe anzeigen lassen, indem Sie zwei zusätzliche Argumente, also insgesamt sechs Argumente, verwenden:

- Das Image-Objekt, das angezeigt werden soll
- Die x-Koordinate
- Die y-Koordinate
- Die Breite
- Die Höhe
- Das Schlüsselwort `this`

Über die zwei zusätzlichen Argumente legen Sie die Breite und Höhe in Pixeln, die das Bild bei der Anzeige haben soll, fest. Falls diese nicht der eigentlichen Größe des Bildes entsprechen, wird das Bild skaliert, um diese Vorgaben zu erfüllen. Hierdurch wird das Bild selbst nicht verändert, so daß Sie diverse Aufrufe von `drawImage()` verwenden können, um ein Image-Objekt in vielen verschiedenen Größen auszugeben.

Zwei Methoden der Image-Klasse sind hilfreich, wenn Sie ein Bild nicht in dessen Originalgröße anzeigen. Die Methode `getHeight()` gibt die Höhe des Bildes und die Methode `getWidth()` die Breite als Integer zurück.

Ein Wort zu Image-Observern

Das letzte Argument der `drawImage()`-Methode ist das Schlüsselwort `this`. Wie Sie sich von den vorigen Tagen her erinnern werden, kann `this` innerhalb eines Objekts verwendet werden, um auf das Objekt selbst zu verweisen.

Über das Schlüsselwort `this` wird in der Methode `drawImage()` angegeben, daß das Applet den Ladevorgang eines Bildes aus dem World Wide Web verfolgen kann. Das Laden von Bildern wird über die `ImageObserver`-Schnittstelle verfolgt. Klassen, die diese Schnittstelle implementieren, wie z.B. `Applet`, können den Ladefortgang eines Bildes verfolgen. Dies ist sehr nützlich, um in einem Programm während des Ladens von Grafikdateien eine Meldung wie »Bilder werden geladen...« anzuzeigen.

Die vorhandene Unterstützung von `ImageObserver` sollte für einfache Anwendungen in bezug auf Bilder in einem Applet ausreichend sein, so daß das Schlüsselwort `this` als Argument für `drawImage()` verwendet wird.

Die Arbeit mit Bildern

Bevor Sie in die Animation von Bildern eintauchen, soll ein einfaches Applet als Beispiel für das Laden eines Bildes von einer URL und die anschließende Anzeige in zwei verschiedenen Größen dienen. Das Applet `Fillmore` in Listing 10.3 zeigt ein Bild des amerikanischen Präsidenten Millard Fillmore.

Listing 10.3: Der gesamte Quelltext von `Fillmore.java`

```

1: import java.awt.Graphics;
2: import java.awt.Image;
3:
4: public class Fillmore extends java.applet.Applet {
5:     Image whig;
6:
7:     public void init() {
8:         whig = getImage(getCodeBase(),
9:             "images/fillmore.jpg");
10:    }
11:
12:    public void paint(Graphics screen) {
13:        int iWidth = whig.getWidth(this);
14:        int iHeight = whig.getHeight(this);
15:        int xPos = 10;
16:        // 25%
17:        screen.drawImage(whig, xPos, 10,
18:            iWidth / 4, iHeight / 4, this);
19:        // 100%
20:        xPos += (iWidth / 4) + 10;
21:        screen.drawImage(whig, xPos, 10, this);
22:    }
23: }
```

Bevor Sie das `Fillmore`-Applet testen können, müssen Sie folgendes tun:

- Erstellen Sie im Ordner `\J21Work` einen Unterordner mit dem Namen `images`.
- Kopieren Sie die Datei `fillmore.jpg` von der CD-ROM des Buches. Als Alternative können Sie jede beliebige `.JPG`-Datei verwenden, die sich bereits auf Ihrem System befindet.
- Erstellen Sie eine Webseite, die das Applet lädt. Das `<APPLET>`-Tag sollte dabei die folgenden Größeneinstellungen vornehmen: `height=400 width=420`. Sie werden eventuell die Attribute `height` und

width anpassen müssen, um im Applet- Fenster ausreichend Platz für das Bild zu schaffen.

Abbildung 10.3 zeigt die Ausgabe des Applets, das die Datei fillmore.jpg in zwei Größen anzeigt: 25 Prozent und 100 Prozent.

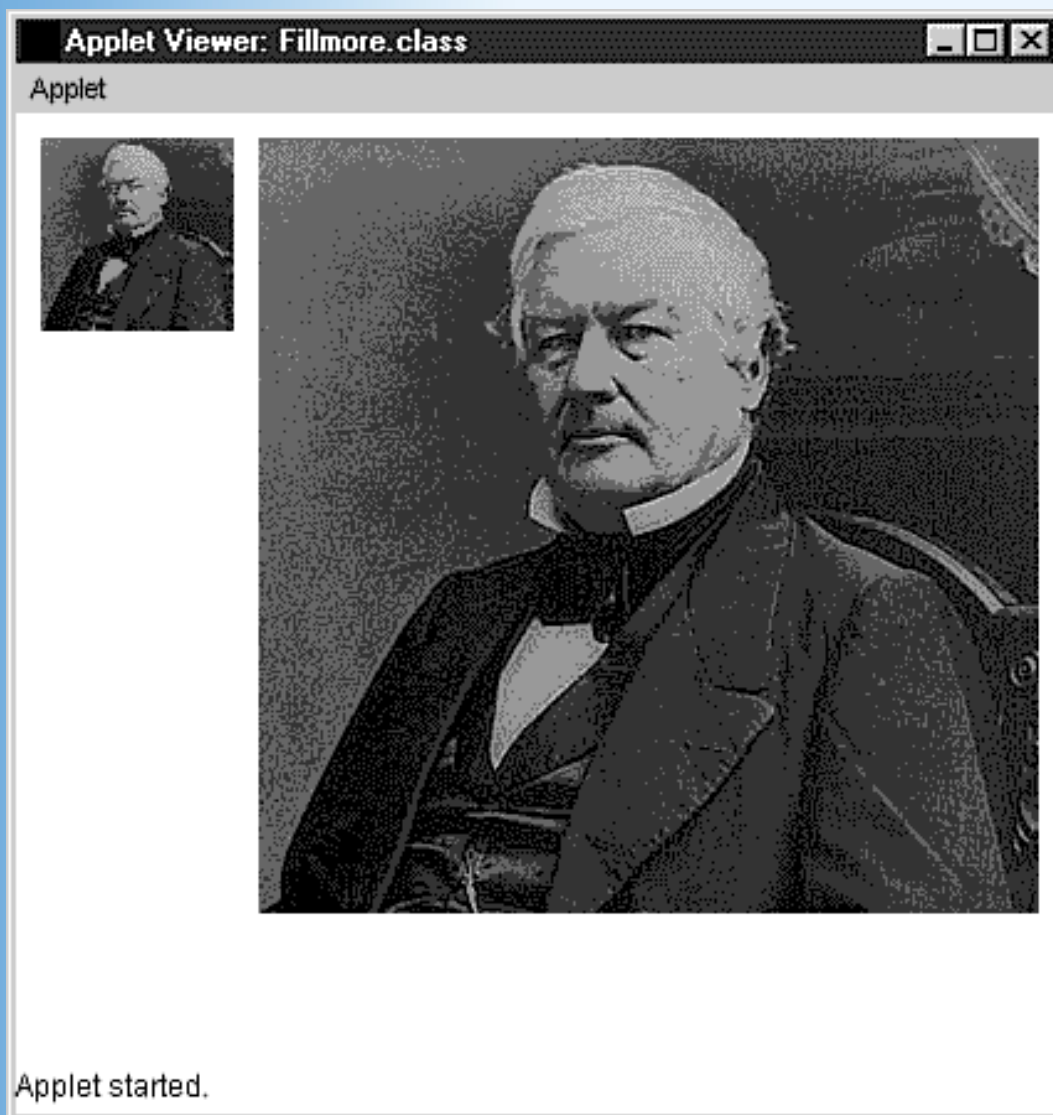


Abbildung 10.3:
Das Fillmore-Applet

In Zeile 5 wird die Variable `whig` der Image-Klasse zugeordnet. Sie müssen die Anweisung `new` hier nicht verwenden, um ein Image-Objekt zu erzeugen, da die `getImage()`- Methode in den Zeilen 8-9 ein solches zurückgibt.

Die Zeilen 13-14 verwenden `getWidth()` und `getHeight()`, zwei Methoden der Klasse `Image`, und speichern die zurückgegebenen Werte in Integer-Variablen. Dies ist notwendig, um eine verkleinerte Version des Bildes in den Zeilen 17-18 zu erzeugen.

In der Zeile 15 wird die `xPos`-Variable definiert, die die x-Koordinate für die beiden Versionen von Präsident Fillmore speichert. In Zeile 20 wird der Wert der Variablen so erhöht, daß sich das große Bild 10 Pixel rechts neben der kleineren Version befindet.

Animationen mit Bildern

Animationen mit Bildern zu erstellen, entspricht vom Prinzip her genau der Animation von Schriften, Farben und anderen Objekten. Sie verwenden dieselben Methoden und dieselbe Vorgehensweise für die Ausgabe, die

Aktualisierung der Ausgabe und zur Reduzierung des Flimmerns. Der einzige Unterschied ist, daß Sie eine Reihe von Bildern haben, die Sie durchwechseln, anstelle einer Reihe von Zeichenoperationen.

Der beste Weg darzulegen, wie man Bilder animiert, ist, ein Beispiel durchzuarbeiten. Das kommende Projekt ist das längste, das Sie bisher in diesem Buch hatten. Keine Angst, es wird detailliert beschrieben. Das Neko-Applet ist eine gute Demonstration für die Programmierung mit Threads, den Umgang mit Bildern und Animation.

Ein Beispiel: Neko

Neko ist eine kleine Macintosh-Animation (ein Spiel), das 1989 von Kenji Gotoh geschrieben wurde. »Neko« heißt auf Japanisch »Katze« und die Animation handelt von einer kleinen Katze, die den Mauszeiger über den Bildschirm jagt, schläft, sich kratzt und sich ansonsten nett verhält. Das Neko-Programm ist seitdem auf fast jede denkbare Plattform übertragen worden und steht auch als Bildschirmschoner zur Verfügung.

Für dieses Beispiel implementieren Sie eine kleine Animation, die auf den Originalgrafiken von Neko basiert. Anders als der Original-Neko, der autonom war (er konnte die Ränder des Fensters »spüren«, sich umdrehen und in eine andere Richtung weiterlaufen), zwingt dieses Applet Neko dazu, von der linken Seite des Fensters aus loszulaufen, in der Mitte zu stoppen, zu gähnen, sich am Ohr zu kratzen, ein bißchen zu schlafen und dann nach rechts weiterzulaufen.

Schritt 1: Bilder zusammenstellen

Ehe Sie mit dem Schreiben des Java-Codes beginnen, um die Animation zu erstellen, sollten Sie alle Bilder zur Verfügung haben, aus der die Animation selbst besteht. Für diese Fassung von Neko werden neun Bilder (die Originalversion verwendet 36) benötigt. Diese sind in Abb. 10.4 zu sehen:



Abbildung 10.4:
Die Bilder für das Neko-Applet

Als Vorbereitung für dieses Projekt kopieren Sie die folgenden neun Bilddateien von der CD-ROM des Buches in den Ordner \J21Work\images, den Sie bereits zuvor erzeugt haben: Awake1.gif, Right1.gif, Right2.gif, Scratch1.gif, Scratch2.gif, Sleep1.gif, Sleep2.gif, Stop.gif und Yawn.gif.

Schritt 2: Organisieren und Laden der Bilder im Applet

Doch nun zum Applet. Die Grundidee ist, daß Sie über einen Satz von Bildern verfügen und diese schnell hintereinander ablaufen lassen, damit der Eindruck einer Bewegung entsteht. Die einfachste Möglichkeit, dies in Java zu erreichen, besteht darin, die Bilder in einem Array von Image-Objekten zu speichern und das jeweils aktuelle Bild dann mit Hilfe einer speziellen Variablen anzuzeigen. Für unser Beispiel soll das Array den Namen nekoPics die Variable currentImage haben. Während die einzelnen Elemente des Arrays mit einer for-Schleife durchlaufen werden, können Sie jedes Mal den Wert des aktuellen Bildes ändern.

Für das Applet Neko erstellen Sie Instanzvariablen, um diese beiden Dinge zu implementieren: ein Array für die einzelnen Bilder mit dem Namen nekoPics und eine Variable des Typs Image namens currentImg, welche das aktuelle Bild für die Anzeige enthält:

```
Image nekoPics[] = new Image[9];
Image currentImg;
```

Das Bild-Array enthält hier neun Elemente, weil die Neko-Animation über neun Bilder verfügt. Wenn Sie einen größeren oder kleineren Satz von Bildern verwenden, müssen Sie die entsprechende Anzahl der Bilder hier angeben.

Da die Neko-Animation die Katzen-/Bilder an verschiedenen Positionen des Bildschirms zeichnet, müssen Sie auch die aktuellen x- und y-Koordinaten verfolgen, damit Sie die verschiedenen Methoden in diesem Applet erkennen können, wo mit dem Zeichnen begonnen werden soll. Der y-Wert bleibt bei diesem Applet konstant (Neko läuft auf immer der gleichen y-Koordinate (50) von links nach rechts), der x-Wert variiert allerdings. Im folgenden werden für diese beiden Positionen zwei Instanzvariablen deklariert:

```
int x;
int y = 50;
```

Doch nun zum Hauptteil des Applet. Während der Initialisierung des Applets werden alle Bilder eingelesen und im Array nekoPics gespeichert. Dazu verwenden Sie einen separaten Aufruf der Methode getImage() für jedes der neun Bilder. Eine etwas weniger redundante Methode ist, ein String-Array mit den Namen der neun Bilddateien zu erzeugen. Dieses Array wird in einer for-Schleife verwendet, um die Dateinamen an die getImage()-Methode zu übergeben. Diese Art Operation läßt sich besonders gut in einer init()-Methode ausführen.

```
public void init() {
    String nekoSrc[] = { "right1.gif", "right2.gif",
        "stop.gif", "yawn.gif", "scratch1.gif",
        "scratch2.gif", "sleep1.gif", "sleep2.gif",
        "awake.gif" };

    for (int i=0; i < nekoPics.length; i++) {
        nekopics[i] = getImage(getCodeBase(),
            "images/" + nekoSrc[i]);
    }
}
```

Da die Bilder in dem Unterordner images gespeichert sind, muß dieser Teil des Pfadargumentes in getImage() sein.

Schritt 3: Bilder animieren

Sobald die Bilder geladen sind, besteht der nächste Schritt darin, die Teile des Applets zu animieren. Dies geschieht innerhalb der run()-Methode des Applets.

In diesem Applet führt Neko fünf wesentliche Aktionen aus:

- Er läuft von der linken Bildschirmseite in das Blickfeld
- Er stoppt in der Mitte und gähnt
- Er kratzt sich viermal
- Er schläft
- Er wacht auf und läuft zur rechten Seite des Bildschirms

Sie könnten dieses Applet zwar so animieren, daß das richtige Bild zur gegebenen Zeit am Bildschirm gezeichnet wird, es ist aber sinnvoller, das Applet so zu schreiben, daß die Aktivitäten von Neko jeweils in einer einzelnen Methode enthalten sind. Auf diese Weise lassen sich bestimmte Aktivitäten (insbesondere die Animation von Nekos Laufen) wieder verwenden, wenn Neko dies in unterschiedlicher Reihenfolge ausführen soll.

Zu Beginn wird eine Methode erstellt, die Neko zum Laufen bringt. Die Methode nekorun() erwartet zwei Argumente: die x-Position des Ausgangspunktes und die x-Position des Endpunktes. Neko läuft dann

zwischen diesen beiden Positionen (der y-Wert bleibt konstant). Hier ist der Anfang der Methode:

```
void nekorun(int start, int end) {
    // noch zu definieren
}
```

Es gibt zwei Bilder, die das Laufen von Neko darstellen und den Effekt des Laufens erzielen: Right1.gif und Right2.gif. Sie müssen also zwischen diesen beiden Bildern wechseln (gespeichert an den Positionen 0 und 1 im Bilder-Array) und diese gleichzeitig über den Bildschirm bewegen. Der Bewegungsteil läßt sich am einfachsten mit einer for-Schleife zwischen Anfangs- und Endargument definieren. Dafür wird die x- Position als aktueller Schleifenwert verwendet.

Um die Bilder zu tauschen, wird geprüft, welches Bild aus dem nekoPics-Array sich aktuell im currentImg-Objekt befindet, und das jeweils andere dann zugewiesen. Dies geschieht bei jeder Wiederholung der for-Schleife.

Durch den Aufruf von repaint() wird das Bild, das sich aktuell in currentImg befindet, ausgegeben.

Als letztes müssen Sie in der nekoRun()-Methode noch dafür sorgen, daß in der for- Schleife vor dem Wechsel der Bilder und der Ausgabe des neuen eine Pause eingelegt wird.

Da jede der Methoden für die Bewegungen von Neko eine Pause benötigt, fügen wir dem Applet eine Methode pause(), die wiederverwendet werden kann, hinzu. Diese Methode verwendet die Methode Thread.sleep(), wie im folgenden gezeigt:

```
void pause(int time) {
    try {
        Thread.sleep(time);
    } catch (InterruptedException e) { }
}
```

Nachdem der Aufruf von pause() eingefügt ist, hat die nekoRun()-Methode die folgende Definition:

```
void nekoRun(int start, int end) {
    for (int i = start; i < end; i+=10) {
        x = i;
        // Bilder tauschen
        if (currentImg == nekoPics[0])
            currentImg = nekoPics[1];
        else currentimg = nekoPics[0];
        repaint();
        pause(150);
    }
}
```

Beachten Sie, daß die Schleife in der zweiten Zeile um 10 Pixel erhöht wird. Warum 10 Pixel und nicht 5 oder 8? Diese Antwort ist überwiegend von Experimenten bestimmt, dabei entdecken Sie, welche Pixel-Anzahl adäquat ist. Zehn ist für diese Animation angemessen. Wenn Sie eigene Animationen erstellen, müssen Sie mit beiden Abständen experimentieren und die Zeit für den Schlaf herausfinden, bis die Animation ihren Vorstellungen entsprechend abläuft.

Sie haben gesehen, daß die nekoRun()-Methode das aktuelle Bild für die Animation in der Variablen currentImg speichert, bevor repaint() aufgerufen wird. Wenden wir uns jetzt der paint()-Methode zu, welche die Einzel./Bilder der Animation ausgibt. In diesem Fall ist die paint()-Methode sehr einfach. paint() ist im wesentlichen verantwortlich für das Zeichnen des aktuellen Bildes an der aktuellen x- und y-Position. Alle diese Informationen werden in Instanzvariablen gespeichert. Doch ehe mit dem Zeichnen begonnen wird, muß sichergestellt sein, daß die Bilder tatsächlich vorhanden sind (diese können auch gerade noch geladen werden). Um dies festzustellen und sich zu vergewissern, daß kein Bild gezeichnet wird, das nicht vorhanden

ist (daraus können die verschiedensten Fehler entstehen), wird ein Test durchgeführt, der sicherstellt, daß `currentimg` nicht null ist, ehe `drawImage()` aufgerufen wird, um das Bild zu zeichnen:

```
public void paint(Graphics screen) {
    if (currentImg != null)
        screen.drawImage(currentImg, x, y, this);
}
```

Im folgenden wird die `run()`-Methode betrachtet, in der sich die wesentlichen Verarbeitungsvorgänge für diese Animation abspielen. Sie haben bereits die `nekorun()`-Methode erstellt; in `run()` rufen Sie nun diese Methode mit den entsprechenden Werten auf, um Neko vom linken Bildschirmrand zur Mitte laufen zu lassen:

```
// Neko läuft vom linken Bildschirmrand zur Mitte
nekoRun(0, size().width / 2);
```



Die `size()`-Methode der Applet-Klasse wurde nach Java 1.02 verworfen. Wenn Sie also dieses Applet für die aktuelle Java-Version 1.2 schreiben wollen, ersetzen Sie einfach die Methode `size()` durch `getSize()`. Die Methode `nekoRun()` würde dann wie folgt aufgerufen werden:

```
nekoRun(0, getSize().width / 2);
```

Das zweitwichtigste Verhalten von Neko in dieser Animation ist das Anhalten und Gähnen. Für alle diese Momente stehen jeweils eigene Bilder (an den Positionen 2 und 3 des Arrays) zur Verfügung, d.h. Sie benötigen keine separaten Methoden, um diese zu zeichnen. Sie müssen lediglich das jeweils passende Bild auswählen, `repaint()` aufrufen und für die Pause die richtige Zeit einstellen. In diesem Beispiel wurde für jede Pause vor dem Anhalten und Gähnen eine Sekunde eingestellt. Die richtige Zeit wurde durch Experimentieren ermittelt. Im folgenden finden Sie den zugehörigen Code:

```
// Stoppen und pausieren
currentImg = nekoPics[2];
repaint();
pause(1000);
// gähnen
currentImg = nekoPics[3];
repaint();
pause(1000);
```

Und nun zum dritten Teil der Animation: Nekos Kratzen. Für diesen Teil ist keine horizontale Bewegung definiert. Sie wechselt zwischen zwei verschiedenen Kratz-Bildern (an den Positionen 4 und 5 des Bilder-Arrays). Da das Kratzen jedoch eine eigene Aktion ist, soll hierfür auch eine eigene Methode verwendet werden (`nekoScratch()`).

Die `nekoscratch()`-Methode enthält ein einziges Argument: die Häufigkeit des Kratzens. Mit diesem Argument können Sie Wiederholungen definieren und innerhalb der Schleife zwischen den beiden verschiedenen Bildern wechseln und diese jeweils neu zeichnen lassen:

```
void nekoScratch(int numTimes) {
    for (int i = numTimes; i > 0; i--) {
        currentImg = nekoPics[4];
        repaint();
        pause(150);
        currentImg = nekoPics[5];
        repaint();
        pause(150);
    }
}
```

```
    }
}
```

Innerhalb der run()-Methode können Sie nekoscratch() mit dem Argument 4 aufrufen:

```
// Viermal kratzen
nekoScratch(4);
```

Weiter geht's! Nachdem sich Neko gekratzt hat, schläft er. Auch hierfür benötigen Sie zwei Bilder (an den Positionen 6 und 7 des Arrays), der Wechsel zwischen den Bildern wiederholt sich jeweils mit einer festgelegten Häufigkeit, gefolgt von einer Pause von 150 Millisekunden. Im folgenden finden Sie die nekosleep()-Methode, die ein einziges Argument erwartet. Dieses Argument gibt an, wie oft die Sequenz wiederholt wird:

```
void nekoSleep(int numTimes) {
    for (int i = numTimes; i > 0; i--) {
        currentImg = nekoPics[6];
        repaint();
        pause(250);
        currentImg = nekoPics[7];
        repaint();
        pause(250);
    }
}
```

Die nekoSleep()-Methode wird in der run()-Methode des Applets mit dem Argument 5 aufgerufen:

```
// 5 "Durchläufe" lang schlafen
nekoSleep(5);
```

Am Ende des Applet wacht Neko auf und läuft zur rechten Seite des Bildschirms. Das Bild für das Aufwachen ist das letzte Bild im Array (nekoPics[8]), und Sie können hierfür erneut die nekorun()-Methode verwenden:

```
// aufwachen und weglaufen
currentImg = nekoPics[8];
repaint();
pause(500);
nekoRun(x, size().width + 10);
```

Schritt 4: Applet fertigstellen

Es gibt noch eine Sache, die zur Fertigstellung des Applets notwendig ist. Die Bilder für die Animation verfügen alle über einen weißen Hintergrund. Wenn Sie diese Bilder auf dem Standardhintergrund von Applets (ein Mittelgrau) zeichnen, entsteht ein nicht sehr attraktives weißes Feld um die einzelnen Bilder.

Um dieses Problem zu beheben, definieren Sie einfach die Hintergrundfarbe des Applets am Anfang der run()-Methode als Weiß:

```
setBackground(Color.white);
```

Dieses Applet enthält viel Code und viele einzelne Methoden, mit denen eine relativ einfache Animation ausgeführt wird, aber im Grunde ist es nicht kompliziert. Der Kern aller Animationen in Java besteht darin, die Einzel./Bilder aufzubauen und dann repaint() aufzurufen, um das Zeichnen am Bildschirm zu ermöglichen.

Beachten Sie, daß in diesem Applet keine Schritte unternommen werden, um den Flimmereffekt zu reduzieren. Es hat sich herausgestellt, daß die Bilder dieses Applets und die Zeichenfläche so klein sind, daß das Flimmern hier nicht zum Problem wird. Wenn Sie eine Animation schreiben, sollten Sie die grundlegenden Dinge zuerst erledigen und dann zusätzliche Verhalten einfügen, um den Ablauf zu optimieren.

Um diesen Abschnitt abzuschließen, zeigt Listing 10.4 den kompletten Code für das Neko-Applet.

Listing 10.4: Der gesamte Quelltext von Neko.java

```

1: import java.awt.Graphics;
2: import java.awt.Image;
3: import java.awt.Color;
4:
5: public class Neko extends java.applet.Applet
6:     implements Runnable {
7:
8:     Image nekoPics[] = new Image[9];
9:     Image currentImg;
10:    Thread runner;
11:    int x;
12:    int y = 50;
13:
14:    public void init() {
15:        String nekoSrc[] = { "right1.gif", "right2.gif",
16:                             "stop.gif", "yawn.gif", "scratch1.gif",
17:                             "scratch2.gif", "sleep1.gif", "sleep2.gif",
18:                             "awake.gif" };
19:
20:        for (int i=0; i < nekoPics.length; i++) {
21:            nekoPics[i] = getImage(getCodeBase(),
22:                                   "images/" + nekoSrc[i]);
23:        }
24:    }
25:
26:    public void start() {
27:        if (runner == null) {
28:            runner = new Thread(this);
29:            runner.start();
30:        }
31:    }
32:
33:    public void stop() {
34:        runner = null;
35:    }
36:
37:    public void run() {
38:        setBackground(Color.white);
39:        // Neko läuft vom linken Bildschirmrand zur Mitte
40:        nekoRun(0, size().width / 2);
41:        // Stoppen und pausieren
42:        currentImg = nekoPics[2];
43:        repaint();
44:        pause(1000);
45:        // gähnen
46:        currentImg = nekoPics[3];
47:        repaint();
48:        pause(1000);
49:        // Viermal kratzen
50:        nekoScratch(4);
51:        // 5 "Durchläufe" lang schlafen

```

```
52:         nekoSleep(5);
53:         // aufwachen und weglaufen
54:         currentImg = nekoPics[8];
55:         repaint();
56:         pause(500);
57:         nekoRun(x, size().width + 10);
58:     }
59:
60:     void nekoRun(int start, int end) {
61:         for (int i = start; i < end; i += 10) {
62:             x = i;
63:             // Bilder tauschen
64:             if (currentImg == nekoPics[0])
65:                 currentImg = nekoPics[1];
66:             else currentImg = nekoPics[0];
67:             repaint();
68:             pause(150);
69:         }
70:     }
71:
72:     void nekoScratch(int numTimes) {
73:         for (int i = numTimes; i > 0; i--) {
74:             currentImg = nekoPics[4];
75:             repaint();
76:             pause(150);
77:             currentImg = nekoPics[5];
78:             repaint();
79:             pause(150);
80:         }
81:     }
82:
83:     void nekoSleep(int numTimes) {
84:         for (int i = numTimes; i > 0; i--) {
85:             currentImg = nekoPics[6];
86:             repaint();
87:             pause(250);
88:             currentImg = nekoPics[7];
89:             repaint();
90:             pause(250);
91:         }
92:     }
93:
94:     void pause(int time) {
95:         try {
96:             Thread.sleep(time);
97:         } catch (InterruptedException e) { }
98:     }
99:
100:    public void paint(Graphics screen) {
101:        if (currentImg != null)
102:            screen.drawImage(currentImg, x, y, this);
103:    }
104: }
```

Wenn dieses Applet mit dem Compiler aus dem JDK 1.2 kompiliert wird, zeigt dieser eine Warnung wegen der verworfenen Methode `size()` an. Diese Warnung können Sie getrost ignorieren - das Applet wird erfolgreich auf Java-1.02- und Java-1.1-kompatiblen Browsern, wie z.B. dem Netscape Navigator, laufen. Eine Java-1.2-Version dieses Applets, Neko12, finden Sie auf der CD-ROM zum Buch im Ordner `\Source\Day10`.

Um dieses Applet zu testen, erzeugen Sie eine Webseite, in dem das Applet-Fenster eine Breite von 300 Pixeln und eine Höhe von 200 Pixeln hat. Abbildung 10.5 zeigt das Ergebnis.



Abbildung 10.5:
Das Neko-Applet

Anmerkung zur Verwendung von Grafik-Kontexten

Wenn Sie ausführlichen Gebrauch von Grafik-Kontexten in Ihren Applets oder Anwendungen machen, sollten Sie sich darüber im klaren sein, daß diese Kontexte häufig bestehen bleiben, nachdem die Arbeit damit abgeschlossen ist, auch wenn keine weiteren Referenzen dazu bestehen. Grafik-Kontexte sind spezielle Objekte im AWT, die im nativen Betriebssystem verwurzelt sind; der Garbage Collector von Java kann diese Grafik-Kontexte nicht selbst entfernen. Da nichtverwendete Objekte die Performance von Java beeinträchtigen können, sollten Sie die `dispose()`-Methode der Klasse `Graphics` verwenden, um Grafik-Kontexte explizit zu löschen. Ein guter Ort für die Platzierung dieser Methode ist die `destroy()`-Methode des Applets (diese haben Sie am 8.Tag als eine der Primärmethoden eines Applets kennengelernt, neben `init()`, `start()` und `stop()`):

```
public void destroy() {
    offscreenGraphics.dispose();
}
```

Doppelte Pufferung

Sie haben bereits eine einfache Möglichkeit kennengelernt, wie sich der Flimmereffekt in Java-Animationen reduzieren läßt. Eine zweite, etwas komplexere, aber auch meistens sinnvollere Technik zur Reduzierung des Flimmereffekts in Java-Animationen besteht in der sogenannten doppelten Pufferung.



Die *doppelte Pufferung* ist ein Vorgang, bei dem alle Zeichenaktivitäten in einem Puffer abseits des Bildschirms vorgenommen werden. Anschließend wird der gesamte Inhalt dieses Puffers in einem Schritt am Bildschirm angezeigt. Diese Technik wird *doppelte Pufferung* genannt, weil es zwei Puffer für Grafikausgaben gibt, zwischen denen Sie wechseln.

Bei der Verwendung der doppelten Pufferung erstellen Sie eine zweite Zeichenfläche (sozusagen außerhalb des Bildschirms), nehmen dort alle Zeichenoperationen vor und zeichnen dann am Ende die gesamte Zeichenfläche in einem Schritt im aktuellen Applet (und damit auf dem Bildschirm). Da sich diese Arbeit hinter den Kulissen vollzieht, wird damit die Möglichkeit ausgeschaltet, daß Zwischenschritte innerhalb des Zeichenvorgangs aus Versehen erscheinen und den Ablauf einer Animation stören.

Die Verwendung der doppelten Pufferung ist nicht immer die beste Lösung. Wenn das Applet viele störende Flimmereffekte aufweist, können Sie auch `update()` überschreiben und nur Teile des Bildschirms neuzeichnen. Dies kann das Problem bereits lösen. Der Doppelpuffer ist weniger effizient als der reguläre Puffer und beansprucht zudem mehr Speicherplatz, in einigen Fällen kann dies also nicht der beste Lösungsansatz sein. Wenn Sie jedoch rigoros alle Flimmereffekte einer Animation entfernen möchten, funktioniert diese Technik ausgesprochen gut.

Um ein Applet zu erstellen, das doppelte Pufferung verwendet, benötigen Sie zweierlei: ein sogenanntes Offscreen-Bild und einen Grafikkontext für dieses Bild. Die beiden simulieren den Effekt der Grafikoberfläche eines Applets: Der Grafikkontext (eine Instanz von `Graphics`) enthält die Zeichenmethoden, wie z.B. `drawImage()` (und `drawString()`), und `Image` enthält die Bildpunkte, die gezeichnet werden sollen.

Um ein Applet mit doppelter Pufferung zu versehen, sind vier weitere wichtige Schritte erforderlich: Zunächst müssen das Offscreen-Bild und der Grafikkontext in Instanzvariablen gespeichert werden, damit diese an die `paint()`-Methode weitergeleitet werden können. Richten Sie in Ihrer Klassendefinition folgende Instanzvariablen ein:

```
Image offscreenImage;
Graphics offscreen;
```

Als zweiten Schritt erstellen Sie während der Initialisierung des Applets ein `Image`- und ein `Graphics`-Objekt und weisen diese jenen Variablen zu (dazu muß die Initialisierung abgewartet werden, damit Sie wissen, wie groß sie werden). Die `createImage()`-Methode gibt Ihnen eine Instanz von `Image`, die Sie dann an die `getGraphics()`-Methode übergeben können, um einen neuen `Graphics`-Kontext für das Bild zu erhalten:

```
offscreenImage = createImage(size().width,
    size().height);
offscreen = offscreenImage.getGraphics();
```

Wann immer Sie jetzt am Bildschirm zeichnen (meist in der `paint()`-Methode), zeichnen Sie nun Offscreen-Grafiken und nicht auf der Zeichenoberfläche des Applets. Um z.B. ein Bild namens `img` an Position 10, 10 zu zeichnen, verwenden Sie diese Zeile:

```
offscreen.drawImage(img, 10, 10, this);
```

Am Ende der `paint`-Methode, wenn alle Zeichnungen im Offscreen-Bild ausgeführt sind, fügen Sie die folgende Zeile ein, um den Offscreen-Puffer auf den tatsächlichen Bildschirm zu übertragen:

```
screen.drawImage(offscreenImage, 0, 0, this);
```

Nun überschreiben Sie noch die Methode `update()`, damit diese den Bildschirm zwischen den Zeichenvorgängen nicht leert:

```
public void update(Graphics g) {
    paint(g);
}
```

Im folgenden werden diese vier Schritte noch einmal zusammengefaßt:

1. Fügen Sie Instanzvariablen für den Bild- und Grafikkontext des Offscreen-Puffers ein.
2. Erstellen Sie ein Image-Objekt und einen Grafikkontext, nachdem das Applet initialisiert ist.
3. Nehmen Sie alle Applet-Zeichnungen im Offscreen-Puffer vor und nicht auf der Zeichenoberfläche des Applets.
4. Am Ende der paint()-Methode zeichnen Sie den Inhalt des Offscreen-Puffers auf den realen Bildschirm.

Das Checkers-Applet

Im folgenden finden Sie ein weiteres Beispiel für eine einfache Animation: Dieses Applet trägt den Namen Checkers. Ein rotes Oval (ein Damestein) bewegt sich von einem schwarzen auf ein weißes Quadrat wie auf einem Damebrett. Am Ende dieser Bewegung kehrt es zum Ausgangspunkt zurück und bewegt sich erneut.

Im folgenden wird erläutert, was dieses Applet ausführt: Die Instanzvariable xpos verfolgt die aktuelle Ausgangsposition des Damesteins (weil er sich horizontal bewegt, bleibt y konstant und muß nicht verfolgt werden, während x sich ändert). In der run()-Methode wird der Wert von x geändert und neu gezeichnet, wobei zwischen jeder Bewegung eine Wartezeit von 100 Millisekunden liegt. Der Damestein bewegt sich von einer Seite des Bildschirms zur anderen und wieder zurück, wobei er wieder seine ursprüngliche Position einnimmt, sobald er auf der rechten Seite des Bildschirms angekommen ist.

In der paint()-Methode werden die Hintergrundquadrate mit der Methode fillRect() gezeichnet (ein weißes und ein schwarzes), und anschließend wird der Damestein mit der Methode fillOval() an seine aktuelle Position gesetzt.

Dieses Applet flimmert ebenso wie das ColorSwirl-Applet sehr stark. Das einfache Überschreiben von update() reicht in diesem Fall nicht aus, da Teile des Bildschirms geleert und neu gezeichnet werden, während sich der Damestein quer über den Bildschirm bewegt. Der Flimmereffekt tritt in diesem Applet vor allem deswegen auf, weil zuerst der Hintergrund und dann darauf der Damestein gezeichnet wird.

Sie könnten dieses Applet dahingehend ändern, daß paint() nur jene Elemente mit clipRect() neu zeichnet, die sich ändern. Auf diese Weise läßt sich das Flimmern reduzieren. Aber bei dieser Strategie müssen die alten und neuen Positionen des Damesteins verfolgt werden, und dies ist nicht sehr elegant. Eine bessere Lösung besteht in diesem Fall darin, den Doppelpuffer einzusetzen und damit alle Flimmereffekte auszuschalten. Für dieses Beispiel einen Doppelpuffer einzufügen ist einfach. Fügen Sie zunächst die Instanzvariablen für das Offscreen-Bild und den Grafik-Kontext ein:

```
Image offscreenImg;
Graphics offscreen;
```

Fügen Sie als zweiten Schritt eine init()-Methode ein, um den Offscreen-Puffer zu initialisieren:

```
public void init() {
    offscreenImg = createImage(size().width, size().height);
    offscreen = offscreenImg.getGraphics();
}
```

Drittens ändern Sie die paint()-Methode, um in den Offscreen-Puffer anstatt in den Haupt-Grafikpuffer zu zeichnen:

```
public void paint(Graphics screen) {
    // Hintergrund zeichnen
```

```

        offscreen.setColor(Color.black);
        offscreen.fillRect(0, 0, 100, 100);
        offscreen.setColor(Color.white);
        offscreen.fillRect(100, 0, 100, 100);
        // Damestein zeichnen
        offscreen.setColor(Color.red);
        offscreen.fillOval(xPos, 5, 90, 90);
        screen.drawImage(offscreenImg, 0, 0, this);
    }

```

Beachten Sie die letzte Anweisung dieser Methode. Dies ist die einzige Anweisung, die direkt auf das Applet etwas ausgibt. Diese Anweisung gibt den gesamten Offscreen- Puffer bei den Koordinaten (0,0) aus. Da offScreenImg in der Größe des Applet-Fensters angelegt wurde, füllt es dieses vollständig aus.

Und schließlich geben Sie in der destroy()-Methode des Applet explizit an, daß der in offscreen gespeicherte Grafik-Kontext entfernt werden soll:

```

public void destroy() {
    offscreen.dispose();
}

```

In Listing 10.5 ist der gesamte Quelltext des Checkers-Applets abgedruckt.

Listing 10.5: Der gesamte Quelltext von Checkers.java

```

1: import java.awt.*;
2:
3: public class Checkers extends java.applet.Applet implements Runnable {
4:     Thread runner;
5:     int xPos = 5;
6:     int xMove = 4;
7:     Image offscreenImg;
8:     Graphics offscreen;
9:
10:
11:     public void init() {
12:         offscreenImg = createImage(size().width, size().height);
13:         offscreen = offscreenImg.getGraphics();
14:     }
15:
16:     public void start() {
17:         if (runner == null); {
18:             runner = new Thread(this);
19:             runner.start();
20:         }
21:     }
22:
23:     public void stop() {
24:         runner = null;
25:     }
26:
27:     public void run() {
28:         Thread thisThread = Thread.currentThread();
29:         while (runner == thisThread) {
30:             xPos += xMove;
31:             if ((xPos > 105) | (xPos < 5))

```

```

32:         xMove *= -1;
33:         repaint();
34:         try {
35:             Thread.sleep(100);
36:         } catch (InterruptedException e) { }
37:     }
38: }
39:
40: public void update(Graphics screen) {
41:     paint(screen);
42: }
43:
44: public void paint(Graphics screen) {
45:     // Hintergrund zeichnen
46:     offscreen.setColor(Color.black);
47:     offscreen.fillRect(0,0,100,100);
48:     offscreen.setColor(Color.white);
49:     offscreen.fillRect(100,0,100,100);
50:     // Damestein zeichnen
51:     offscreen.setColor(Color.red);
52:     offscreen.fillOval(xPos,5,90,90);
53:     screen.drawImage(offscreenImg, 0, 0, this);
54: }
55:
56: public void destroy() {
57:     offscreen.dispose();
58: }
59: }

```

Sie können dieses Applet auf einer Webseite testen, indem Sie im <APPLET>-Tag für die Größe die folgenden Attribute verwenden: height=200 und width=300. Das Ergebnis sehen Sie in Abbildung 10.6.

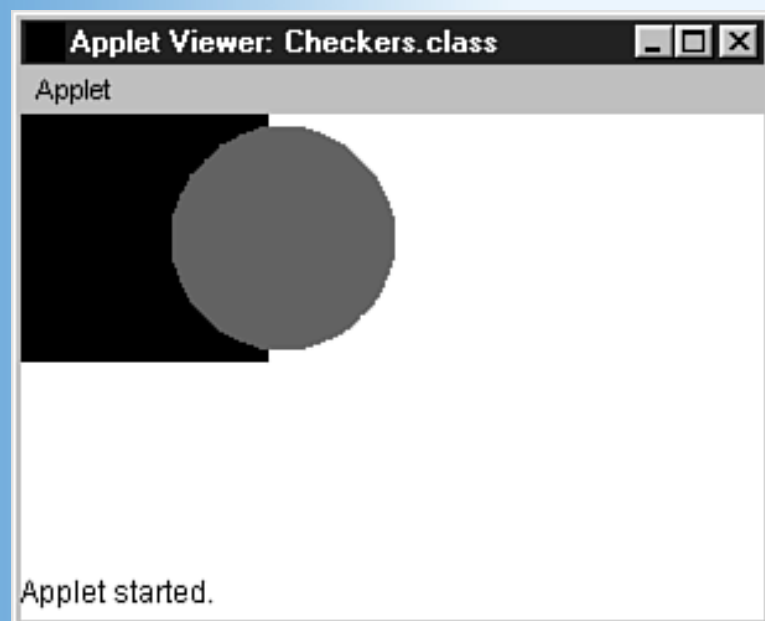


Abbildung 10.6:
Das Checkers-Applet

Klänge laden und verwenden

Java bietet eine vordefinierte Unterstützung für das Abspielen von Klängen in Verbindung mit dem Ablauf von Animationen oder zum eigenständigen Abspielen an. Ebenso wie die Unterstützung für Bilder befindet sich auch die Unterstützung für Klänge in den Klassen Applet und AWT. Die Verwendung von Klängen ist also ebenso einfach wie das Laden und Verwenden von Bildern.

Vor Java 1.2 wurde nur ein Klangformat unterstützt: 8 kHz Mono AU mit mu-law-Codierung von Sun. AU-Dateien sind kleiner als andere Klangdateien in anderen Formaten, aber die Tonqualität ist nicht besonders gut. Wenn Sie Sounds nutzen wollten, die in anderen Formaten vorlagen, mußten Sie diese in das AU-Format konvertieren, was oft mit einem Qualitätsverlust verbunden war.

Java 1.2 bietet eine weitaus umfassendere Audio-Unterstützung. Sie können digitalisierte Klänge der folgenden Formate laden und abspielen: AIFF, AU und WAF. Zusätzlich werden drei Formate auf MIDI-Basis unterstützt: Typ 0 MIDI, Typ 1 MIDI und RMF. Die stark erweiterte Unterstützung von Klängen kann mit Audio Daten in acht- und 16 Bit, Mono oder Stereo, und Sampling-Raten von 8 kHz bis 48 kHz umgehen.

Die einfachste Möglichkeit, einen Klang zu laden und abzuspielen, bietet die `play()`-Methode. Diese bildet einen Teil der Applet-Klasse und steht deshalb in Applets für Sie zur Verfügung. Die `play()`-Methode ist der `getImage()`-Methode sehr ähnlich. Auch sie kann in folgenden beiden Formen verwendet werden:

- `play()` mit einem Argument, ein URL-Objekt, lädt und spielt den an dieser URL angegebenen Audio-Clip ab.
- `play()` mit zwei Argumenten, eine Basis-URL und eine Pfadangabe, lädt und spielt diese Audiodatei ab. Das erste Argument ist sinnvollerweise ein Aufruf von `getDocumentBase()` oder `getCodeBase()`.

Die folgende Codezeile lädt beispielsweise die Datei `meow.au` und spielt den darin enthaltenen Klang ab. Die Datei befindet sich im Verzeichnis `audio`, welches wiederum im selben Verzeichnis wie das Applet plaziert ist:

```
play(getCodeBase(), "audio/meow.au");
```

Die `play()`-Methode lädt die Klangdatei und spielt den Ton sobald wie möglich ab, nachdem der Aufruf erfolgt ist. Wenn der Klang nicht gefunden wird, erscheint keine Fehlermeldung, der Ton ist dann nur einfach nicht zu hören.

Wenn Sie einen bestimmten Klang wiederholt abspielen möchten, starten und stoppen Sie die Klangdatei oder führen diese als Schleife aus (um sie immer wieder abzuspielen). In diesem Fall verwenden Sie die Applet-Methode `getAudioClip()`, um die Klangdatei in eine Instanz der `AudioClip`-Klasse (Teil von `java.applet`) zu laden. Vergessen Sie nicht, diese zu importieren. Im Anschluß daran können Sie direkt mit diesem `AudioClip`-Objekt arbeiten.

Angenommen, Sie haben eine Klangschleife erstellt, die permanent im Hintergrund des Applets ausgeführt werden soll. Im Initialisierungscode können Sie folgende Zeile für eine solche Klangdatei verwenden:

```
AudioClip clip = getAudioClip(getCodeBase(),
"audio/loop.wav");
```

Die Methode `getAudioClip()` kann nur in einem Applet aufgerufen werden. Unter Java 1.2 können Applikationen Sound-Dateien über die Methode `newAudioClip()` der Klasse `Applet` laden. Im Anschluß wird das vorige Beispiel umgeschrieben für die Verwendung in einer Applikation:

```
AudioClip clip = newAudioClip("audio/loop.wav");
```

Um den Clip einmal abzuspielen, verwenden Sie die `play()`-Methode:

```
clip.play();
```


Um einen aktuell ablaufenden Soundclip anzuhalten, verwenden Sie die stop()-Methode:

```
clip.stop();
```

Um für den Clip eine Schleife zu definieren (ihn wiederholt abzuspielen), verwenden Sie die loop()-Methode:

```
clip.loop();
```

Wenn die Methode getAudioClip() oder newAudioClip() den angegebenen Klang nicht findet oder diesen aus einem bestimmten Grund nicht laden kann, wird null zurückgegeben. Es ist sinnvoll, den Code für diesen Fall zu testen, ehe Sie die Klangdatei abzuspielen versuchen, da ein versuchter Aufruf der play()-, stop()- und loop()- Methoden für ein null-Objekt einen Fehler zur Folge hat (eine Ausnahme).

In einem Applet lassen sich beliebig viele Klangdateien abspielen; alle Klänge werden genau so miteinander vermischt, wie sie im Applet abgespielt werden.

Beachten Sie, daß bei der Verwendung von Hintergrundklängen mit Wiederholungsschleifen die Klangdatei nicht automatisch angehalten wird, wenn der Thread des Applets gestoppt wird. Das heißt, wenn ein Leser zu einer anderen Seite wechselt, wird der Klang des ersten Applets weiterhin abgespielt. Sie können dieses Problem lösen, indem Sie den Hintergrundklang des Applets mit der stop()-Methode anhalten:

```
public void stop() {
    if (runner != null) {
        if (bgSound != null)
            bgSound.stop();
        runner.stop();
        runner = null;
    }
}
```

Listing 10.6 zeigt eine einfache Grundstruktur für ein Applet, das zwei Klänge abspielt: Der erste, ein Hintergrundklang namens loop.au, wird wiederholt abgespielt. Der zweite, ein Piepsignal (beep.au), wird alle fünf Sekunden abgespielt. (Auf das Bild für dieses Applet habe ich verzichtet, denn es zeigt ausschließlich einen einfachen String am Bildschirm.)

Listing 10.6: Applet, das Klänge abspielt.

```
1: import java.awt.Graphics;
2: import java.applet.AudioClip;
3:
4: public class AudioLoop extends java.applet.Applet
5:     implements Runnable {
6:
7:     AudioClip bgSound;
8:     AudioClip beep;
9:     Thread runner;
10:
11:     public void start() {
12:         if (runner == null) {
13:             runner = new Thread(this);
14:             runner.start();
15:         }
16:     }
17:
18:     public void stop() {
19:         if (runner != null) {
20:             if (bgSound != null)
21:                 bgSound.stop();
```

```

22:         runner = null;
23:     }
24: }
25:
26: public void init() {
27:     bgSound = getAudioClip(getCodeBase(), "loop.au");
28:     beep = getAudioClip(getCodeBase(), "beep.au");
29: }
30:
31: public void run() {
32:     if (bgSound != null)
33:         bgSound.loop();
34:     Thread thisThread = Thread.currentThread();
35:     while (runner == thisThread) {
36:         try {
37:             Thread.sleep(5000);
38:         } catch (InterruptedException e) { }
39:         if (beep != null)
40:             beep.play();
41:     }
42: }
43:
44: public void paint(Graphics screen) {
45:     screen.drawString("Playing Sounds ...", 10, 10);
46: }
47: }

```

Um das AudioLoop-Applet zu testen, erzeugen Sie eine Webseite mit einem Applet- Fenster, das eine Höhe von 100 Pixeln und eine Breite von 200 Pixeln hat. Die Audio Dateien loop.au und beep.au sollten Sie von der CD-ROM zum Buch in den Ordner \J21Work auf Ihrem System kopieren. Wenn Sie das Applet ausführen, ist ein String die einzige visuelle Ausgabe. Sie sollten zwei verschiedene Klänge hören, während das Applet läuft.

Die init()-Methode in den Zeilen 26 und 29 lädt die beiden Klangdateien loop.au und beep.au. Hier wurde kein Versuch unternommen, sicherzustellen, daß Dateien auch tatsächlich wie erwartet geladen werden. Es besteht also die Möglichkeit, daß die Instanzvariablen bgSound und beep den Wert null haben, wenn die jeweilige Datei nicht geladen werden kann. In diesem Fall könnte die start()-Methode oder eine beliebige andere Methode nicht aufgerufen werden. Es sollte daher an anderer Position im Applet ein Test dafür ausgeführt werden.

Ein entsprechender Test wurde deshalb an anderen Stellen eingefügt, nämlich in der run()-Methode in den Zeilen 32 und 39. Hier werden die Methoden loop() und play() für die AudioClip-Objekte aufgerufen - allerdings nur, wenn die Variablen bgSound und beep einen anderen Wert als null enthalten.

Schließlich sollten Sie einen Blick auf Zeile 20 werfen, welche den Hintergrundklang explizit abschaltet, wenn der Thread angehalten wird. Da das Abspielen von Hintergrundklängen nicht automatisch mit dem Beenden des Thread aufhört, muß dies explizit eingefügt werden.

Zusammenfassung

Heute haben Sie einiges über diverse Methoden gelernt, die Sie verwenden und überschreiben können - start(), stop(), paint(), repaint(), run() und update() - und Sie haben eine der elementaren Grundlagen für die Erzeugung und Verwendung von Threads kennengelernt. Sie haben ebenfalls gelernt, wie Sie Bilder in Ihren

Applets verwenden -, das Auffinden und Laden von Bildern und die Verwendung der Methode `drawImage()`, um Bilder auszugeben und zu animieren.

Eine Animationstechnik, die Sie jetzt verwenden können, ist die doppelte Pufferung, die Flimmern in Ihren Animationen praktisch völlig eliminiert, allerdings auf Kosten der Effizienz und der Geschwindigkeit. Über ein `Image`-Objekt und einen `Graphics`- Kontext können Sie einen Offscreen-Puffer erzeugen, auf den Sie zeichnen. Das Ergebnis der ganzen Operationen wird zuletzt auf dem Bildschirm angezeigt.

Sie haben gelernt, Klänge zu verwenden, die Sie in Ihre Applets integrieren können, wann immer Sie diese benötigen - in bestimmten Situationen oder als Hintergrund- Sound, der wiederholt abgespielt wird, solange das Applet läuft. Sie haben gelernt, wie Sie Klänge sowohl mit der Methode `play()` als auch mit der Methode `getAudioClip()` auffinden, laden und abspielen.

Fragen und Antworten

Frage:

Im Neko-Programm fügen Sie den Ladevorgang für die Bilder in die `init()`-Methode ein. Mir scheint, Java benötigt eine sehr lange Zeit für das Laden der Bilder, und da `init()` nicht im Haupt-Thread des Applet liegt, findet hier eine deutliche Pause statt. Warum läßt sich der Ladevorgang nicht am Anfang der `run()`-Methode einfügen?

Antwort:

Hinter den Kulissen spielen sich auch noch andere Dinge ab. Die `getImage()`- Methode lädt das Bild nämlich nicht wirklich, sondern gibt beinahe unmittelbar ein `Image`-Objekt zurück, damit während der Initialisierung keine langen Verarbeitungszeiten anfallen. Die Bilddaten, auf die `getImage()` verweist, werden nicht geladen, solange das Bild nicht benötigt wird. Auf diese Art muß Java keine riesigen Bilder im Arbeitsspeicher aufbewahren, wenn das Programm nur ein kleines Stück davon benötigt. Statt dessen bleibt lediglich die Referenz auf diese Daten erhalten, während das Laden der notwendigen Bereiche später stattfindet.

Frage:

Ich habe das Neko-Applet kompiliert und ausgeführt. Dabei geschehen merkwürdige Dinge: Die Animation beginnt in der Mitte und läßt Einzel./Bilder aus. Es scheint, also ob nur einige Bilder geladen worden sind, wenn das Applet ausgeführt wird.

Antwort:

Ja, genau das ist der Fall. Da das Laden der Bilder das Bild nicht tatsächlich lädt, animiert das Applet sozusagen den leeren Bildschirm, während die Bilder noch geladen werden. Das Applet scheint dann in der Mitte zu starten, Einzel./Bilder zu erstellen und überhaupt nicht zu funktionieren.

Für dieses Problem gibt es drei Lösungen. Die erste besteht darin, eine Animationsschleife zu verwenden (d.h. von vorne zu beginnen, sobald es angehalten wird). Eventuell werden die Bilder dann geladen, und die Animation funktioniert korrekt. Als zweite Lösung, die allerdings nicht sehr gut ist, können Sie eine kleine Pause vor der Ausführung definieren, damit die Bilder geladen werden können, ehe die Animation ausgeführt wird. Die dritte und beste Möglichkeit ist, `ImageObserver` zu verwenden, mit deren Hilfe sich sicherstellen läßt, daß kein Bereich der Animation abgespielt wird, ehe nicht die notwendigen Bilder geladen sind. Nähere Erläuterungen hierzu erhalten Sie in der Dokumentation zur Schnittstelle `ImageObserver`.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 1

Tag 11

Einfache Benutzeroberflächen für Applets

Durch die Popularität des Apple Macintosh und von Microsoft Windows erwarten die meisten Computerbenutzer heute, daß deren Software eine grafische Benutzeroberfläche bietet und mit der Maus bedient werden kann.

Diese Annehmlichkeiten der Software sind zwar benutzerfreundlich in vielen Sprachen, aber programmiererunfreundlich. Software für grafische Benutzeroberflächen zu schreiben, kann für einen neuen Entwickler eine größere Herausforderung sein.

Glücklicherweise hat Java diesen Prozeß mit dem Abstract Windowing Toolkit vereinfacht. Das Abstract Windowing Toolkit - AWT - besteht aus einer Reihe von Klassen, mit denen sich grafische Benutzeroberflächen erstellen und benutzen lassen.

Heute werden Sie das AWT zur Erstellung einer Benutzeroberfläche für ein Applet verwenden. Sie werden lediglich die Techniken von Java 1.02 verwenden, da dies die Standardversion der Sprache bei den Applet-Anwendern geblieben ist.

Morgen werden Sie lernen, wie Sie die einzelnen Komponenten auf einer Benutzerschnittstelle anordnen. An Tag 13 vervollständigen Sie eine Benutzerschnittstelle, indem Sie sie auf die Eingaben des Benutzers reagieren lassen.

Nachdem Sie gelernt haben, wie Sie Programme mit dem Abstract Windowing Toolkit erstellen, sind Sie bereit, Techniken von Java 1.2 für die Erstellung von Applikationen an den Tagen 20 und 21 zu verwenden.

Das Abstract Windowing Toolkit

Das *Abstract Windowing Toolkit*, auch AWT genannt, ist ein Satz von Klassen, der es Ihnen ermöglicht, eine grafische Benutzeroberfläche zu erstellen und Eingaben des Benutzers über die Maus und die Tastatur entgegenzunehmen.

Da Java eine plattformunabhängige Sprache ist, haben die Benutzerschnittstellen, die mit dem AWT entworfen werden, auf allen Systemen die gleiche Funktionalität und abgesehen von den Plattformeigenheiten die gleiche Erscheinung.



Eines werden Sie lernen, wenn Sie Java-Applets mit dem AWT erstellen, nämlich, daß manche Dinge nicht gänzlich konsistent über alle Plattformen hinweg sind. Die verschiedenen Java-Laufzeitumgebungen, die von Netscape, Microsoft und anderen Firmen für deren Browser erstellt wurden, sind sich nicht immer einig, wie eine AWT- Benutzerschnittstelle funktionieren sollte. Es ist sehr

wichtig, daß Sie ein Applet mit einer Benutzerschnittstelle auf möglichst vielen Plattformen und in möglichst vielen Browsern testen.

In bezug auf das AWT besteht eine Benutzerschnittstelle aus drei Dingen:

- *Komponenten*. Alles, was in eine Benutzerschnittstelle eingefügt werden kann, darunter anklickbare Schaltflächen, scrollbare Listen, Pop-up-Menüs, Kontrollkästchen und Textfelder.
- *Container*. Eine Komponente, die andere Komponenten beinhalten kann. Mit einer dieser Komponenten haben Sie bereits die ganze Zeit gearbeitet - dem Applet-Fenster. Andere Vertreter wären Panels, Dialogfelder und Fenster.
- *Layout-Manager*. Ein Objekt, das festlegt, wie die Komponenten in einem Container arrangiert werden. Sie sehen den Layout-Manager in einer Benutzerschnittstelle nicht, aber Sie sehen auf alle Fälle das Ergebnis seiner Arbeit.

Die Klassen des AWT befinden sich alle in dem Paket `java.awt`. Um alle diese Klassen in einem Programm verfügbar zu machen, können Sie die folgende Anweisung verwenden:

```
import java.awt.*;
```

Diese Anweisung importiert alle Komponenten, Container und Layout-Manager, die Sie für den Entwurf Ihrer Benutzerschnittstelle verwenden können. Sie können natürlich auch einzelne import-Anweisungen für die Klassen, die Sie in Ihrem Programm verwenden, einsetzen.

Die Klassen des AWT sind, wie alle anderen Teile der Java-Klassenbibliothek, in einer Vererbungshierarchie angeordnet. Wenn Sie lernen, wie Sie mit einer bestimmten Klasse umgehen, lernen Sie auch, wie Sie mit anderen Klassen umgehen, die von dieser Superklasse abgeleitet wurden.

Die Basiskomponenten der Benutzeroberfläche

Komponenten werden auf einer Benutzerschnittstelle angeordnet, indem Sie in einen Container eingefügt werden. Ein Container ist seinerseits eine Komponente, so daß er in einen anderen Container eingefügt werden kann. Sie werden diese Funktionalität nutzen, wenn Sie beginnen, mit Layout-Managern zu arbeiten, um eine Benutzerschnittstelle zu arrangieren.

Am einfachsten demonstriert man den Entwurf einer Benutzerschnittstelle mit einem Container, den Sie bereits die ganze Zeit verwenden - die Klasse `Applet`.

Einfügen von Applet-Komponenten

Eine Komponente wird über die folgenden zwei Schritte in einen Container eingefügt:

- Erzeugen der Komponente
- Aufruf der `add()`-Methode des Containers mit der Komponente

Da alle Applets Container sind, können Sie die `add()`-Methode eines Applets verwenden, um eine Komponente direkt in das Applet-Fenster einzufügen.

Jede Komponente einer AWT-Benutzerschnittstelle ist eine Klasse. Eine Komponente erzeugen Sie aus diesem Grund, indem Sie ein Objekt dieser Klasse erstellen.

Die Klasse `Button` repräsentiert anklickbare Schaltflächen in einer Benutzerschnittstelle. Sie können eine Schaltfläche erzeugen, indem Sie die Beschriftung der Schaltfläche deren Konstruktor als Parameter übergeben:

```
Button panic = new Button("Panic!");
```

Diese Anweisung erstellt ein Button-Objekt, das mit dem Text "Panic!" beschriftet ist.

Nachdem Sie eine Komponente erstellt haben, stellt der Aufruf der `add()`-Methode eines Containers mit der Komponente als Argument die einfachste Möglichkeit dar, die Komponente in diesen Container einzufügen.

Da ein Applet ein Container ist, kann die folgende Anweisung in einem Applet verwendet werden, um das `panic`-Objekt in ein Applet-Fenster einzufügen:

```
add(panic);
```

Das Hinzufügen einer Komponente sorgt nicht unmittelbar dafür, daß diese angezeigt wird. Statt dessen erscheint sie erst, wenn die `paint()`-Methode ihres Containers aufgerufen wird. Dies erledigt Java hinter den Kulissen. Sie können allerdings den Aufruf von `paint()` in einem Applet erzwingen, indem Sie dessen `repaint()`-Methode aufrufen.

Wenn Sie eine Komponente in einen Container einfügen, geben Sie keine x,y-Koordinaten an, die festlegen, wo die Komponente platziert werden soll. Die Platzierung der Komponenten übernimmt der Layout-Manager des Containers.



Morgen werden Sie mehr über Layout-Manager lernen. Das Standard-Layout für einen Container platziert die einzelnen Komponenten in einer Zeile von links nach rechts, bis nicht mehr ausreichend Platz vorhanden ist. In diesem Fall wird in der nächsten Zeile mit der Platzierung der Komponenten fortgefahren. Dies wird als *Flow Layout* bezeichnet und von der Klasse `FlowLayout` umgesetzt.

Der beste Ort in einem Applet für die Erstellung von Komponenten ist die Methode `init()`. Dies soll ein Applet mit einer Schaltfläche in Listing 11.1 demonstrieren. Das Applet `Slacker` erzeugt ein Button-Objekt und fügt es in das Applet-Fenster ein. Die Schaltfläche wird angezeigt, sobald die `paint()`-Methode - geerbt von der Klasse `Applet` - aufgerufen wird.

Listing 11.1: Der komplette Quelltext von `Slacker.java`

```
1: import java.awt.*;
2:
3: public class Slacker extends java.applet.Applet {
4:     String note = "I am extremely tired and would prefer not " +
5:         "to be clicked. Please interact somewhere else.";
6:     Button tired = new Button(note);
7:
8:     public void init() {
9:         add(tired);
10:    }
11: }
```

Testen Sie dieses Applet auf einer Seite mit dem folgenden `<APPLET>`-Tag:

```
<APPLET CODE="Slacker.class" WIDTH=550 HEIGHT=75>
</APPLET>
```

Abbildung 11.1 zeigt das Ergebnis im Applet-Viewer.

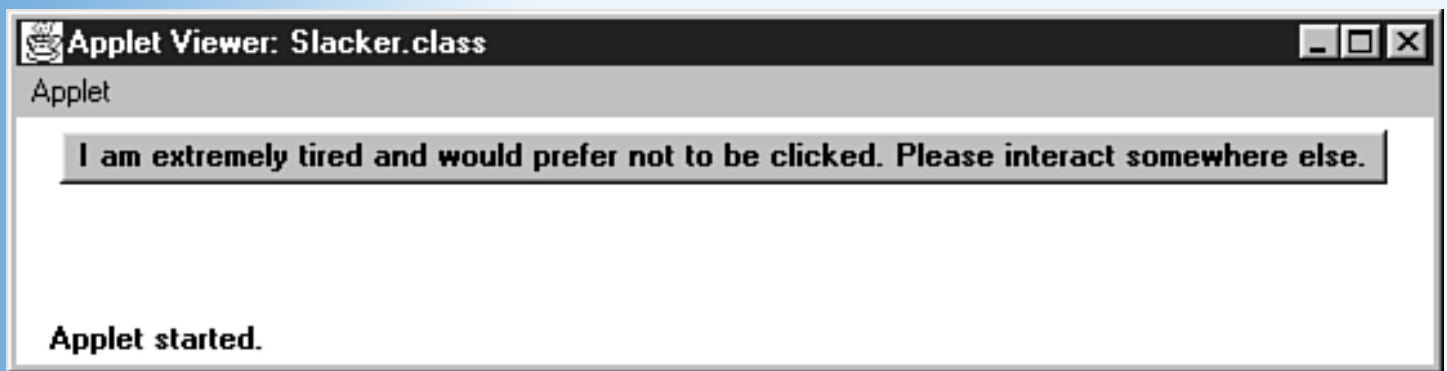


Abbildung 11.1:
Das Slacker-Applet

Labels

Die einfachste Komponente der Benutzeroberfläche ist ein Label, das mit der Label- Klasse erzeugt wird. Im wesentlichen ist ein Label ein String, der zur Beschriftung anderer Komponenten der Benutzeroberfläche verwendet wird. Labels sind nicht direkt durch den Benutzer editierbar.

Im Vergleich zu normalem Text (den Sie unter Verwendung von `drawString()` in der `paint()`-Methode zeichnen) weist ein Label folgende Vorteile auf:

- Sie müssen ein Label nicht selbst in `paint()` nachzeichnen. Labels sind ein Element im AWT, und das AWT übernimmt für Sie diese Aufgabe.
- Labels passen sich dem Layout des Layout-Managers des Containers an, in dem sie sich befinden, anstatt an eine bestimmte x,y-Koordinate gebunden zu sein, wie das bei einem über `drawString()` ausgegebenen String der Fall ist.

Um ein Label zu erstellen, verwenden Sie einen der folgenden Konstruktoren:

- `Label()` erzeugt ein leeres Label für links ausgerichteten Text.
- `Label(String)` erzeugt einen Label mit einem bestimmten String, der ebenfalls links ausgerichtet ist.
- `Label(String, int)` erzeugt ein Label mit einem bestimmten String und einer bestimmten Ausrichtung. Die verfügbaren Ausrichtungen sind in Klassenvariablen in Label gespeichert und leicht zu behalten: `Label.RIGHT`, `Label.LEFT` und `Label.CENTER`.

Sie können die Schrift des Labels mit der Methode `setFont()` ändern, die Sie an Tag 9 kennengelernt haben. Diese Methode kann entweder für den Container des Labels (z.B. ein Applet), was dann alle Komponenten in diesem Container betrifft, oder nur für das Label selbst aufgerufen werden. Mit folgendem einfachen Code werden ein paar Labels in *Helvetica Bold* erstellt. Abbildung 12.3 zeigt, wie diese Labels auf dem Bildschirm aussehen.

Die Methode `setText(String)` eines Labels kann dafür verwendet werden, den Text eines Labels nach dessen Erstellung zu ändern. Der neue Text, der mit String übergeben wird, wird angezeigt, sobald die Komponente neu gezeichnet wird. Sie können auch die `getText()`-Methode verwenden, um den aktuellen Text des Labels zu ermitteln.

Listing 11.2 zeigt ein einfaches Applet, das ein paar Labels mit *Helvetica Bold* erstellt.

Listing 11.2: Der komplette Quelltext von Labels.java

```
1: import java.awt.*;
2:
3: public class Labels extends java.applet.Applet {
4:     Label lefty = new Label("Bleeding heart!");
```



```

5:      Label center = new Label("Centrist!", Label.CENTER);
6:      Label righty = new Label("Hardliner!", Label.RIGHT);
7:      Font lf = new Font("Helvetica", Font.BOLD, 14);
8:      GridLayout layout = new GridLayout(3,1);
9:
10:     public void init() {
11:         setFont(lf);
12:         setLayout(layout);
13:         add(lefty);
14:         add(center);
15:         add(righty);
16:     }
17: }

```

Testen Sie dieses Applet auf einer Seite mit dem folgenden <APPLET>-Tag:

```

<APPLET CODE="Labels.class" WIDTH=150 HEIGHT=175>
</APPLET>

```

In Abbildung 11.2 sehen Sie die Ausgabe des Applets im Applet-Viewer. Dies ist ein gutes Tool für diesen Zweck, da Sie die Größe des Fensters verändern können. Dabei können Sie beobachten, wie die drei Labels neu ausgerichtet werden. Das Label »Hardliner!« klebt an der rechten Ecke des Applet-Fensters und das Label »Centrist!« bleibt zentriert.



Abbildung 11.2:
Das Labels-Applet

Die Zeilen 8 und 12 dieses Applets werden verwendet, um ein GridLayout-Objekt zu erstellen und dieses Objekt als Layout Manager für den Container zu verwenden. Dieser Layout-Manager wird später behandelt. Er wird hier allerdings benötigt, da Labels in dem Standard-Layout-Manager für Container - FlowLayout - nicht ausgerichtet werden. Hier werden auch die einzelnen Komponenten in einem Raster mit einer Spalte und drei Zeilen angeordnet.

Schaltflächen

Anklickbare Schaltflächen können mit der Button-Klasse erzeugt werden, wie Sie das bereits bei dem Slacker-Applet gesehen haben. Schaltflächen sind in einer Benutzerschnittstelle sehr nützlich, um Aktionen auszulösen. Die Schaltfläche Beenden könnte z.B. ein Programm beenden.

Um eine Schaltfläche zu erstellen, benutzen Sie einen der folgenden Konstruktoren:

- `Button()` erzeugt eine leere Schaltfläche ohne Beschriftung.
- `Button(String)` erzeugt eine Schaltfläche mit dem angegebenen String als Beschriftung.

Nachdem Sie ein Schaltflächenobjekt erstellt haben, können Sie den Wert für seine Beschriftung mit der `getLabel()`-Methode ermitteln und die Beschriftung mit der `setLabel(String)`-Methode setzen.

Listing 11.3 zeigt das VCR-Applet, das einige vertraute Kommandos auf Schaltflächen präsentiert.

Listing 11.3: Der komplette Quelltext von VCR.java

```

1: import java.awt.*;
2:
3: public class VCR extends java.applet.Applet {
4:     Button rewind = new Button("Rewind");
5:     Button play = new Button("Play");
6:     Button ff = new Button("Fast Forward");
7:     Button stop = new Button("Stop");
8:     Button eat = new Button("Eat Tape");
9:
10: public void init() {
11:     add(rewind);
12:     add(play);
13:     add(ff);
14:     add(stop);
15:     add(eat);
16: }
17: }
```

Testen Sie dieses Applet auf einer Seite mit dem folgenden `<APPLET>`-Tag:

```

<APPLET CODE="VCR.class" WIDTH=300 HEIGHT=60>
</APPLET>
```

Abbildung 11.3 zeigt dieses Applet im Applet-Viewer. Beachten Sie bitte, daß sich die Schaltfläche `Eat Tape` in einer neuen Zeile befindet, da in der vorangegangenen Zeile kein Platz mehr für sie war. Wenn Sie das Applet-Fenster 500 Pixel anstatt 300 Pixel breit gemacht hätten, würden alle fünf Schaltflächen in einer Zeile angeordnet werden.

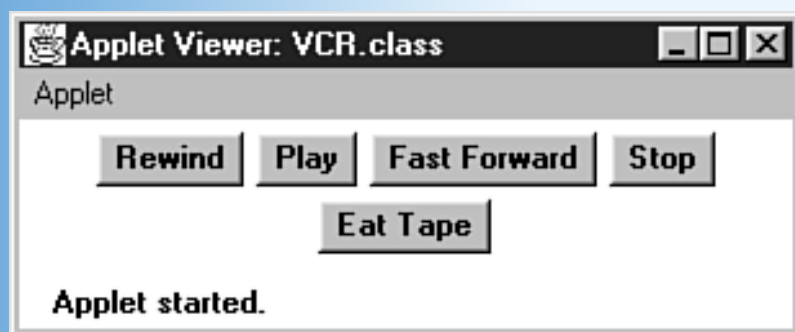


Abbildung 11.3:
Das VCR-Applet

Kontrollfelder

Kontrollfelder sind beschriftete oder unbeschriftete Komponenten einer Benutzeroberfläche, die zwei Status haben: ein und aus (oder angekreuzt und nicht angekreuzt, gewählt und nicht gewählt, true und false usw.). Typischerweise werden diese Komponenten dazu verwendet, bestimmte Optionen in einem Programm zu wählen bzw. diese Auswahl aufzuheben, wie das z.B. bei Bildschirmschonern unter Windows mit dem Kontrollkästchen Kennwortschutz der Fall ist (siehe auch Abbildung 11.4).

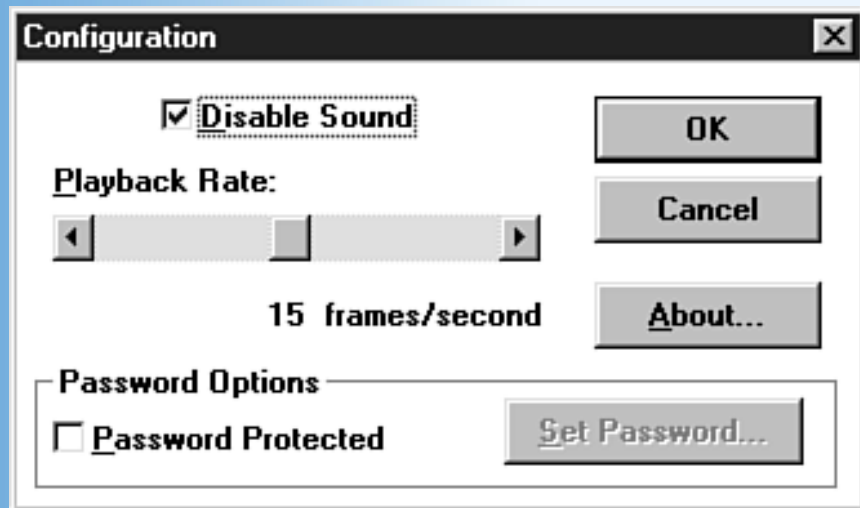


Abbildung 11.4:
Ein Dialogfeld mit Kontrollkästchen

Kontrollkästchen schließen sich normalerweise nicht gegenseitig aus, d.h. wenn Sie z.B. fünf Kontrollkästchen in einem Container haben, können alle zur selben Zeit markiert oder nicht markiert sein.

Diese Komponente kann auch als Kontrollfeldgruppe organisiert werden. In dieser Form werden Kontrollkästchen auch als Optionsfelder bezeichnet. In einer solchen Gruppe kann immer nur ein Optionsfeld markiert sein.

Beide Arten von Kontrollfeldern werden mit der Klasse `Checkbox` erzeugt. Mit den folgenden Konstruktoren erzeugen Sie Kontrollfelder, die sich gegenseitig nicht ausschließen:

- `Checkbox()` erzeugt ein leeres Kontrollfeld, das nicht ausgewählt ist.
- `Checkbox(String)` erzeugt ein Kontrollfeld mit dem angegebenen String als Beschriftung.

Nachdem Sie ein `Checkbox`-Objekt erzeugt haben, können Sie die Methode `setState(boolean)` verwenden, um den Status zu setzen. Wenn Sie `true` als Argument übergeben, wird das Kontrollfeld markiert. Übergeben Sie dagegen `false`, heben Sie die Markierung auf. Die Methode `getState()` gibt einen booleschen Wert zurück, der den aktuellen Status des Kontrollfeldes angibt.

Fünf Kontrollfelder werden in Listing 11.4 erstellt. Dieses Applet ermöglicht es Ihnen, bis zu fünf Berühmtheiten zu wählen, die tschechischer Abstammung sind. Nur eines der Kontrollfelder ist anfänglich markiert - Model/Schauspielerin Paulina Porizkova.

Listing 11.4: Der komplette Quelltext von `CheckACzech.java`

```
1: import java.awt.*;
2:
3: public class CheckACzech extends java.applet.Applet {
4:     Checkbox c1 = new Checkbox("Milos Forman");
5:     Checkbox c2 = new Checkbox("Paulina Porizkova");
```

```

6:      Checkbox c3 = new Checkbox("Ivan Reitman");
7:      Checkbox c4 = new Checkbox("Tom Stoppard");
8:      Checkbox c5 = new Checkbox("Ivana Trump");
9:
10:     public void init() {
11:         add(c1);
12:         c2.setState(true);
13:         add(c2);
14:         add(c3);
15:         add(c4);
16:         add(c5);
17:     }
18: }

```

In Abbildung 11.5 sehen Sie die Ausgabe des Applets, das mit dem folgenden <APPLET> -Tag in einer Webseite getestet werden kann:

```

<APPLET CODE="CheckACzech.class" WIDTH=150 HEIGHT=200>
</APPLET>

```



Abbildung 11.5:
Das CheckACzech-Applet

Um mehrere Kontrollfelder in eine Gruppe einzubringen, in der nur eines zur selben Zeit markiert sein kann, wird ein CheckboxGroup-Objekt mit einer Anweisung wie der folgenden erzeugt:

```
CheckboxGroup radio = new CheckboxGroup();
```

Das CheckboxGroup-Objekt überwacht alle Optionsfelder in seiner Gruppe. Sie verwenden dieses Objekt als Extra-Argument im Checkbox-Konstruktor.

Checkbox(String, CheckboxGroup, boolean) erzeugt ein Optionsfeld, das mit dem übergebenen String beschriftet ist. Das Optionsfeld wird dem CheckboxGroup-Objekt, das als Argument übergeben wurde, hinzugefügt. Als drittes Argument wird true übergeben, wenn das Optionsfeld markiert sein soll, ansonsten false.



Unter Java 1.2 wurde der Konstruktor `Checkbox(String, CheckboxGroup, boolean)` als *deprecated* ausgewiesen, d.h. es ist eine bessere Methode verfügbar. Wenn Sie kein Applet mit Java 1.02 schreiben, dann sollten Sie den Konstruktor `Checkbox(String, boolean, CheckboxGroup)` verwenden. Die Anwendung ist dieselbe, lediglich das zweite und das dritte Argument sind vertauscht.

Das folgende Beispiel erzeugt eine Gruppe mit zwei dazugehörigen Optionsfeldern:

```
CheckboxGroup betterDarrin = new CheckboxGroup();
Checkbox r1 = new Checkbox("Dick York", betterDarrin, true);
Checkbox r2 = new Checkbox("Dick Sargent", betterDarrin, false);
```

Das Objekt `betterDarrin` wird zur Gruppierung der Optionsfelder `r1` und `r2` verwendet. Das `r1`-Objekt, das die Beschriftung "Dick York" trägt, ist ausgewählt. Nur ein Mitglied der Gruppe kann zur selben Zeit ausgewählt sein. Aus diesem Grund ist es nicht möglich, daß das dritte Argument sowohl für `r1` als auch für `r2` `true` ist.

Wenn Sie versuchen, `true` auf mehr als ein Optionsfeld in einer Gruppe anzuwenden, wird nur das letzte markiert. In einer Gruppe muß keines der Optionsfelder markiert sein.

Listing 11.5 demonstriert eine Optionsfeld-Gruppe. Das Applet zeigt eine Gruppe mit Optionsfeldern für fünf Entertainer polnischer Abstammung und wählt einen davon aus - Krzysztof Kieslowski, der Regisseur von *Blue*, *White* und *Red*.

Listing 11.5: Der komplette Quelltext von `PickAPole.java`

```
1: import java.awt.*;
2:
3: public class PickAPole extends java.applet.Applet {
4:     CheckboxGroup p = new CheckboxGroup();
5:     Checkbox p1 = new Checkbox("Samuel Goldwyn", p, false);
6:     Checkbox p2 = new Checkbox("Krzysztof Kieslowski", p, true);
7:     Checkbox p3 = new Checkbox("Klaus Kinski", p, false);
8:     Checkbox p4 = new Checkbox("Joanna Pacula", p, false);
9:     Checkbox p5 = new Checkbox("Roman Polanski", p, false);
10:
11:     public void init() {
12:         add(p1);
13:         add(p2);
14:         add(p3);
15:         add(p4);
16:         add(p5);
17:     }
18: }
```

Mit dem folgenden `<APPLET>`-Tag in einer Webseite kann das Applet getestet werden. Das Ergebnis sehen Sie in Abbildung 11.6:

```
<APPLET CODE="PickAPole.class" WIDTH=150 HEIGHT=200>
</APPLET>
```

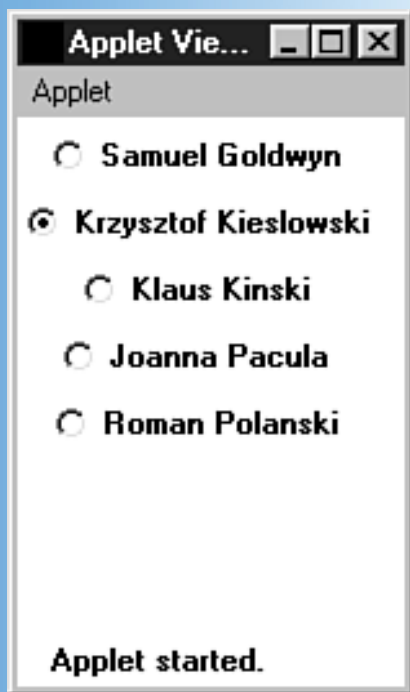


Abbildung 11.6:
Das PickAPole-Applet

Mit der Methode `setCurrent(Checkbox)` können Sie in einer Optionsfeldgruppe das aktuell ausgewählte Optionsfeld festlegen. Außerdem gibt es die Methode `getCurrent()`, die das aktuell ausgewählte Optionsfeld zurückgibt.

Kombinationslistenfeld

Listenfelder, die mit der Klasse `Choice` erzeugt werden, sind Komponenten, die es ermöglichen, einen einzelnen Eintrag aus einem Listenfeld auszuwählen. Sie treffen diese Listenfelder oft in Formularen auf Webseiten an. Abbildung 11.7 zeigt ein Beispiel dieser Listenfelder, das von der Personal Bookshelf Website von Pearson Technology Group Publishing stammt.

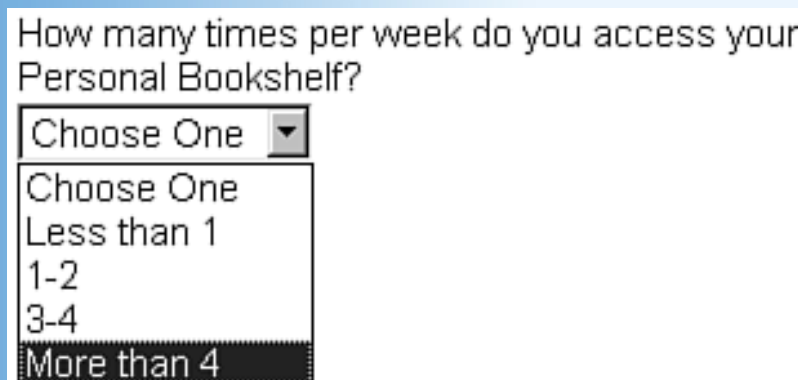


Abbildung 11.7:
Ein Beispiel für Listenfelder

Der erste Schritt bei der Erstellung eines Listenfeldes ist die Erzeugung eines `Choice`- Objekts, das die Liste aufnimmt, wie das in der folgenden Anweisung gezeigt wird:

```
Choice gender = new Choice();
```

Einzelne Elemente werden dem Kombinationslistenfeld über die Methode `addItem(String)` des Objekts

hinzugefügt. Die folgenden Anweisungen fügen zwei Einträge in das Kombinationslistenfeld `gender` ein:

```
gender.addItem("Male");
gender.addItem("Female");
```

Sie können auch, nachdem Sie das Kombinationslistenfeld in einen Container eingefügt haben, weitere Elemente der Liste hinzufügen.



Die Methode `addItem(String)` wurde in den Java-Versionen nach 1.02 verworfen. Verwenden Sie statt dessen die Methode `add(String)`, wenn Sie Programme für eine der nachfolgenden Versionen entwerfen.

Nachdem Sie das Kombinationslistenfeld erstellt haben, wird es wie jede andere Komponente auch in einen Container eingefügt. Dafür verwenden Sie die `add()`-Methode des Containers mit dem Kombinationslistenfeld als Argument.

In Listing 11.6 sehen Sie ein Applet, das den Trend fortsetzt, mit Java internationale Entertainer zu erkennen. Das Applet `SelectASpaniard` erstellt ein Listenfeld mit Berühmtheiten spanischer Abstammung, in dem ein Eintrag ausgewählt werden kann.

Listing 11.6: Der komplette Quelltext von `SelectASpaniard.java`

```
1: import java.awt.*;
2:
3: public class SelectASpaniard extends java.applet.Applet {
4:     Choice span = new Choice();
5:
6:     public void init() {
7:         span.addItem("Pedro Almodóvar");
8:         span.addItem("Antonio Banderas");
9:         span.addItem("Charo");
10:        span.addItem("Xavier Cugat");
11:        span.addItem("Julio Iglesias");
12:        add(span);
13:    }
14: }
```

Dieses Applet können Sie mit dem folgenden HTML-Tag testen. Das Ergebnis sehen Sie in Abbildung 11.8:

```
<APPLET CODE="SelectASpaniard.class" WIDTH=150 HEIGHT=200>
</APPLET>
```

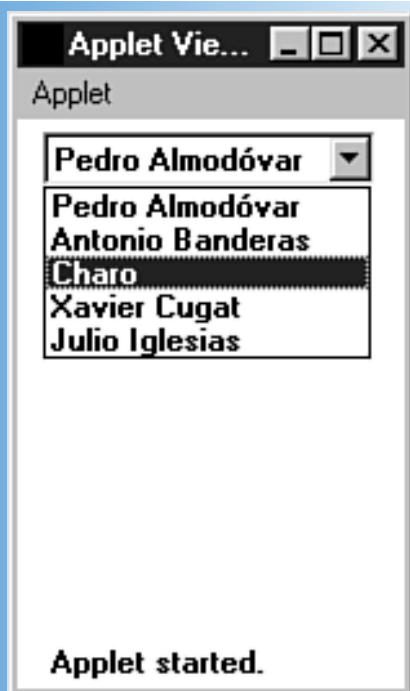


Abbildung 11.8:
Das SelectASpaniard-Applet

Die Klasse Choice bietet einige Methoden, mit denen Sie ein Listenfeld kontrollieren können:

- Die Methode `getItem(int)` gibt den Text des Listeneintrags der mit dem Integer-Argument übergebenen Indexposition an. Wie bei Arrays hat das erste Element den Index 0, das zweite den Index 1 usw.
- Die Methode `countItems()` gibt die Anzahl der Elemente in der Liste zurück. Diese wurde in Java 1.2 verworfen und durch die Methode `getItemCount()` ersetzt, die dieselbe Aufgabe erfüllt.
- Die `getSelectedIndex()`-Methode gibt den Index des aktuell ausgewählten Eintrags in der Liste zurück.
- Die Methode `getSelectedItem()` gibt den Text des aktuell ausgewählten Elements zurück.
- Die Methode `select(int)` wählt den Eintrag mit dem angegebenen Index.
- Die Methode `select(String)` wählt den ersten Eintrag in der Liste, der mit dem angegebenen Text übereinstimmt.

Textfelder

Früher in diesem Kapitel haben Sie Labels für Text verwendet, der vom Benutzer nicht verändert werden kann. Mit einem Textfeld erzeugen Sie eine Komponente für editierbaren Text. Textfelder werden mit der Klasse `TextField` erstellt.

Um ein Textfeld zu erstellen, benutzen Sie einen der folgenden Konstruktoren:

- `TextField()` erzeugt ein leeres Textfeld, ohne eine Größe vorzugeben.
- `TextField(int)` erzeugt ein leeres Textfeld, das genügend Platz bietet, um die mit dem Integer-Argument angegebene Anzahl von Zeichen aufzunehmen. Setzen Sie diese Methode nur für Java 1.02 ein; in 1.2 wurde sie durch `TextField(String, int)` ersetzt.
- `TextField(String)` erzeugt ein mit dem angegebenen String initialisiertes Textfeld, ohne eine Größe vorzugeben.
- `TextField(String, int)` erzeugt ein mit dem angegebenen String initialisiertes Textfeld, das genügend Platz bietet, um die mit dem Integer-Argument angegebene Anzahl von Zeichen aufzunehmen.

Das width-Attribut eines Textfeldes ist unter einem Layout-Manager relevant, der die Größe der Komponenten nicht verändert, wie das z.B. beim FlowLayout-Layout-Manager der Fall ist. Sie werden mehr Erfahrung hiermit bekommen, wenn Sie morgen mit Layout-Managern arbeiten.

Die folgende Anweisung erzeugt ein leeres Textfeld, das genügend Platz für 30 Zeichen bietet:

```
TextField name = new TextField(30);
```

Mit der folgenden Anweisung wird ein Textfeld namens name erzeugt, das den Text "Puddin N. Tane" enthält.

```
TextField name = new TextField("Puddin N. Tane", 30);
```

Sie können auch ein Textfeld erstellen, das die Zeichen, die eingetippt werden, mit allgemeinen Zeichen verbirgt. Diese Textfeldart wird häufig verwendet, um ein eingegebenes Paßwort vor neugierigen Augen zu verstecken.

Um das Zeichen festzulegen, mit dem die Zeichen in dem Textfeld verborgen werden sollen, wird in Java 1.02 die Methode setEchoCharacter(char) der Klasse TextField verwendet (in den Folgeversionen von Java sollte die Methode setEchoChar(char) verwendet werden). Wenn ein Literal benutzt wird, um das Zeichen festzulegen, sollte es in einfache Anführungszeichen eingeschlossen werden, wie z.B. '*'. Java interpretiert jedes Literal in doppelten Anführungszeichen als String-Objekt.

Das folgende Beispiel erzeugt ein Textfeld und legt das Pfund-Zeichen (#) als Zeichen fest, das angezeigt wird, wenn Text in das Textfeld eingegeben wird:

```
TextField passkey = new TextField(16);
passkey.setEchoCharacter('#');
```

Das Applet in Listing 11.7 erzeugt einige Textfelder. Labels werden zur Kennzeichnung der einzelne Felder verwendet - normalerweise verwenden Sie Labels hierfür, anstatt einen erklärenden Text in das Textfeld einzufügen. Eines der Textfelder verwendet ein Zeichen, um den eingegebenen Text zu verbergen.

Listing 11.7: Der komplette Quelltext von OutOfSite.java

```
1: import java.awt.*;
2:
3: public class OutOfSite extends java.applet.Applet {
4:     Label siteLabel = new Label("Site Name: ");
5:     TextField site = new TextField(25);
6:     Label addressLabel = new Label("Site Address: ");
7:     TextField address = new TextField(25);
8:     Label passwordLabel = new Label("Admin Password: ");
9:     TextField password = new TextField(25);
10:
11:     public void init() {
12:         add(siteLabel);
13:         add(site);
14:         add(addressLabel);
15:         add(address);
16:         add(passwordLabel);
17:         password.setEchoCharacter('*');
18:         add(password);
19:     }
20: }
```


Testen Sie dieses Applet auf einer Webseite mit dem folgenden <APPLET>-Tag:

```
<APPLET CODE="OutOfSite.class" WIDTH=350 HEIGHT=125>
</APPLET>
```

Da dieses Applet den Standard-Layout-Manager verwendet, entscheidet nur die Breite des Applet-Fensters darüber, ob die sechs Komponenten sich in verschiedenen Zeilen befinden. Je nach Plattform, die Sie verwenden, kann es sein, daß Sie die Breite des Applet-Fensters anpassen müssen, um eine mit Abbildung 11.9 vergleichbare Ausgabe zu erhalten. (Im nächsten Kapitel lernen Sie, wie Sie Layout-Manager verwenden, um dieses Problem zu vermeiden.)

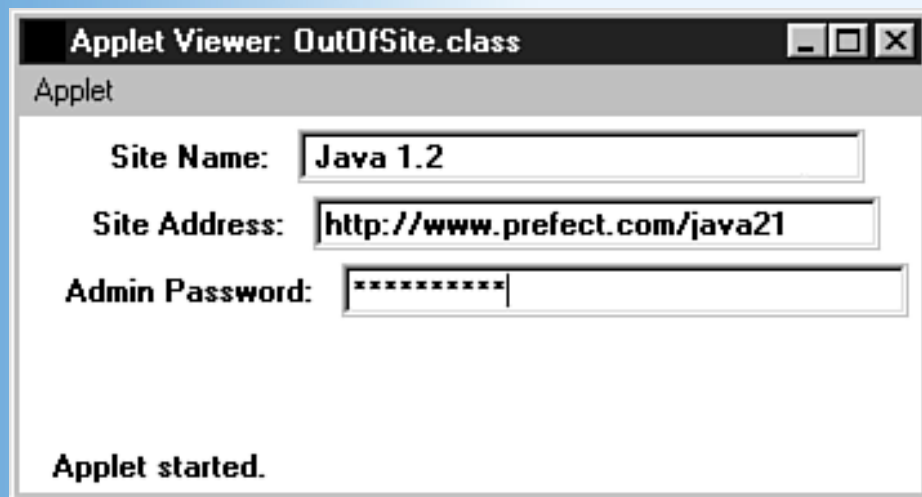


Abbildung 11.9:
Das OutOfSite-Applet

Die Klasse `TextField` bietet einige Methoden, mit denen sich Textfelder steuern lassen:

- Die Methode `getText()` gibt den Text des Feldes (als `String`) zurück.
- Die Methode `setText(String)` füllt das Textfeld mit dem angegebenen Text.
- Die `setEditable(boolean)`-Methode legt fest, ob das Textfeld editiert werden kann. `false`, als Argument übergeben, verhindert, daß ein Textfeld editiert wird, und `true` (der Standardwert) ermöglicht das Editieren des Textes im Textfeld.
- Die Methode `isEditable()` gibt einen booleschen Wert zurück, der angibt, ob das Textfeld editiert werden kann (`true`) oder nicht (`false`).

Mehrzeilige Textfelder

Mehrzeilige Textfelder (engl. `Textareas`), die mit der Klasse `TextArea` erzeugt werden, sind editierbare Textfelder, die mehr als nur eine Zeile Text verarbeiten können. Mehrzeilige Textfelder verfügen über horizontale und vertikale Bildlaufleisten, die es dem Benutzer ermöglichen, durch den Text zu scrollen, der sich in der Komponente befindet.

Um ein mehrzeiliges Textfeld zu erstellen, verwenden Sie einen der folgenden Konstruktoren:

- `TextArea()` erzeugt ein leeres Textfeld, dessen Höhe und Breite nicht vorgegeben sind.
- `TextArea(int, int)` erzeugt ein leeres Textfeld mit der vorgegebenen Anzahl von Zeilen (erstes Argument) und der angegebenen Anzahl Breite in Zeichen (zweites Argument).
- `TextArea(String)` erzeugt ein Textfeld, das den angegebenen String enthält und dessen Höhe und Breite nicht vorgegeben sind.
- `TextArea(String, int, int)` erzeugt ein Textfeld, das den angegebenen String enthält, mit der

vorgegebenen Anzahl von Zeilen (erstes Argument) und der angegebenen Anzahl Breite in Zeichen (zweites Argument).

Das Applet in Listing 11.8 zeigt ein mehrzeiliges Textfeld an, das beim Start des Programms mit einem String gefüllt wird.

Listing 11.8: Der komplette Quelltext von OutOfSite.java

```

1: import java.awt.*;
2:
3: public class Virginia extends java.applet.Applet {
4:     String letter = "Dear Editor:\n" +
5:         "I am 8 years old.\n" +
6:         "Some of my little friends say there is no Santa Claus." +
7:         " Papa\n" +
8:         "says, ''If you see it in The Sun it's so.'' Please tell" +
9:         " me the truth,\n" +
10:        "is there a Santa Claus?\n\n" +
11:        "Virginia O'Hanlon\n" +
12:        "115 West 95th Street\n" +
13:        "New York";
14:    TextArea lt;
15:
16:    public void init() {
17:        lt = new TextArea(letter, 10, 50);
18:        add(lt);
19:    }
20: }
```

Testen Sie das Virginia-Applet auf einer Webseite mit dem folgenden <APPLET>-Tag:

```
<APPLET CODE="Virginia.class" WIDTH=450 HEIGHT=250>
</APPLET>
```

Die Darstellung des Applets im Applet-Viewer sehen Sie in Abbildung 11.10.

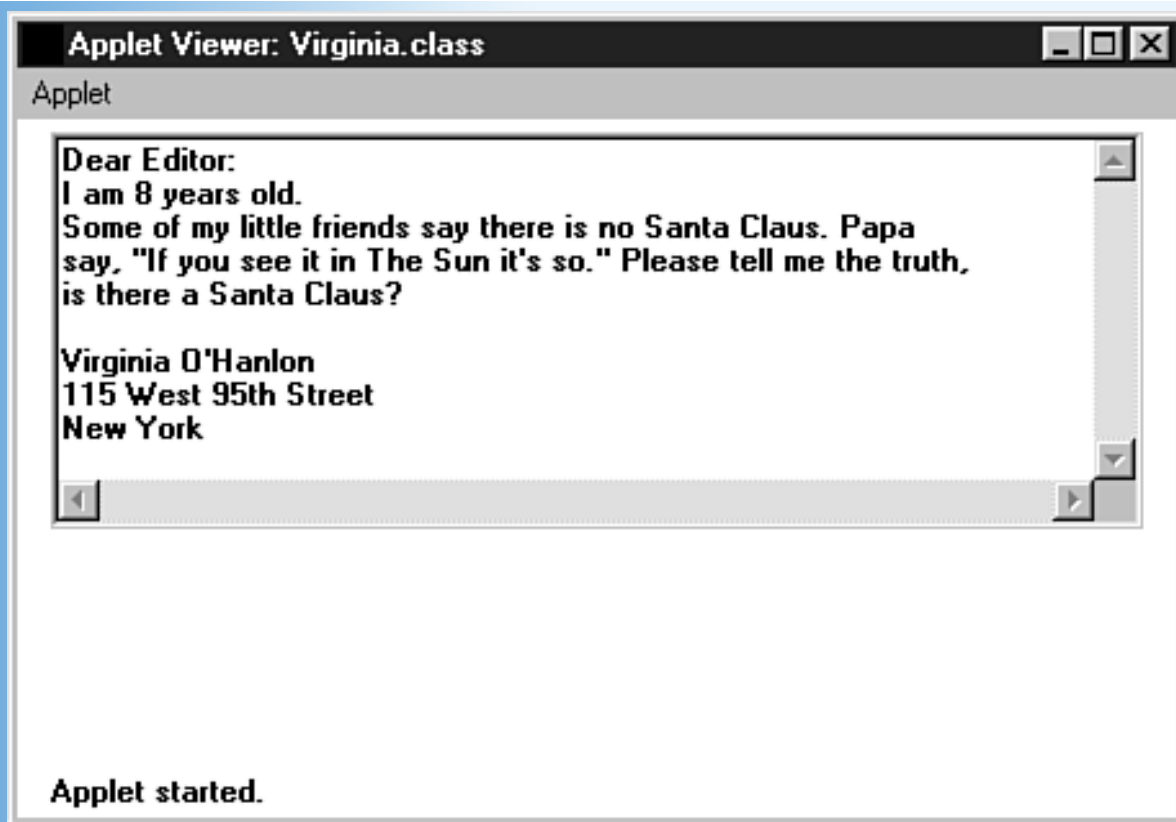


Abbildung 11.10:
Das Virginia-Applet

Sowohl einzeilige als auch mehrzeilige Textfelder werden von der Klasse `TextComponent` abgeleitet. Aus diesem Grund können viele der Verhaltensweisen von einzeiligen Textfeldern auch in mehrzeiligen Textfeldern verwendet werden. Die Methoden `setText()`, `getText()`, `setEditable()` und `isEditable()` stehen für mehrzeilige Textfelder ebenfalls zur Verfügung. Außerdem können Sie die folgenden Methoden verwenden:

- Die Methode `insertText(String, int)` fügt den übergebenen String an der Position mit dem Zeichenindex ein, der durch den Integer angegeben ist. Der Index beginnt beim ersten Zeichen mit 0 und wird dann hochgezählt. Diese Methode wurde in den Java-Versionen nach 1.02 verworfen und mit der Methode `insert(String, int)` ersetzt.
- Die Methode `replaceText(String, int, int)` ersetzt den Text zwischen den mit den Integer-Argumenten übergebenen Positionen durch den Text, der als erstes Argument übergeben wird. Auch diese Methode wurde in den Java-Versionen nach 1.02 verworfen und durch die Methode `replace(String, int)` ersetzt.

Listenfelder

Listenfelder, die mit der Klasse `List` erstellt werden, ähneln den Kombinationslistenfeldern mit zwei wesentlichen Unterschieden:

- Ein Listenfeld kann so konfiguriert werden, daß mehr als ein Eintrag zugleich ausgewählt ist.
- Listenfelder öffnen sich nicht, wenn sie angeklickt werden, wie das bei Kombinationslistenfeldern der Fall ist. Statt dessen werden die einzelnen Einträge ähnlich wie in einem mehrzeiligen Textfeld angezeigt. Wenn die Liste mehr Einträge enthält, als angezeigt werden, kann mit einer Bildlaufleiste durch die einzelnen Einträge geblättert werden.

Ein Listenfeld erstellen Sie, indem Sie ein Objekt der Klasse `List` erzeugen und anschließend der Liste einzelne Einträge hinzufügen. Die Klasse `List` besitzt die folgenden Konstruktoren:

- `List()` erzeugt ein leeres Listenfeld, das nur die gleichzeitige Auswahl eines Eintrages zulässt.
- `List(int, boolean)` erzeugt ein Listenfeld, das die angegebene Anzahl von Einträgen gleichzeitig anzeigen kann. Diese kann durchaus kleiner als die gesamte Anzahl der Einträge in der Liste sein. Der boolesche Parameter gibt an, ob mehrere Einträge gleichzeitig ausgewählt werden können (`true`) oder nicht (`false`).

Nachdem Sie ein `List`-Objekt erzeugt haben, verwenden Sie dessen `addItem(String)`-Methode, um Einträge in die Liste einzufügen. (Unter Java 1.2 wurde diese Methode durch die Methode `add(String)` ersetzt.)

Das folgende Beispiel erzeugt ein Listenfeld und fügt zwei Einträge ein:

```
List lackeys = new List();
lackeys.addItem("Rosencrantz");
lackeys.addItem("Guildenstern");
```

Nachdem eine Liste erzeugt und Einträge hinzugefügt wurden, sollte die Liste mit der `add()`-Methode in ihren Container eingefügt werden. In Listing 11.9 wird die Erstellung von einem Listenfeld mit sieben Einträgen gezeigt.

Listing 11.9: Der komplette Quelltext von `Hamlet.java`

```
1: import java.awt.*;
2:
3: public class Hamlet extends java.applet.Applet {
4:     List hm = new List(5, true);
5:
6:     public void init() {
7:         hm.addItem("Hamlet");
8:         hm.addItem("Claudius");
9:         hm.addItem("Gertrude");
10:        hm.addItem("Polonius");
11:        hm.addItem("Horatio");
12:        hm.addItem("Laertes");
13:        hm.addItem("Ophelia");
14:        add(hm);
15:    }
16: }
```

Die Ausgabe des Applets sehen Sie in Abbildung 11.11. In der HTML-Seite wurde das folgende `<APPLET>`-Tag verwendet.

```
<APPLET CODE="Hamlet.class" WIDTH=200 HEIGHT=150>
</APPLET>
```

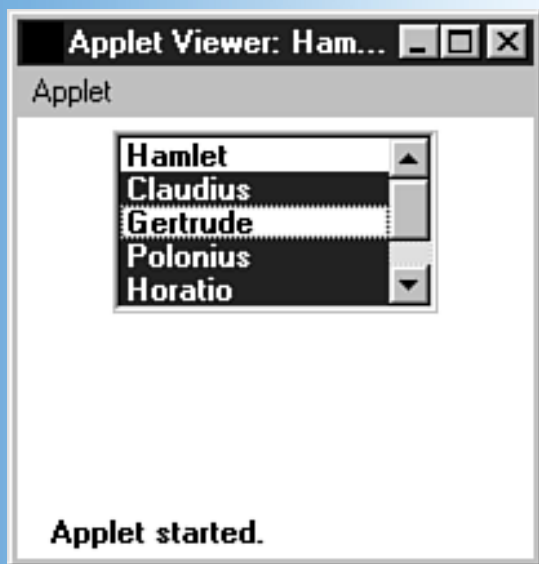


Abbildung 11.11:
Das Hamlet-Applet - Claudius, Polonius und Horatio sind ausgewählt

Listenfelder besitzen einige Methoden, die genauso funktionieren wie die entsprechenden Methoden bei Kombinationslistenfeldern: `getItem(int)`, `countItems()`, `getSelectedIndex()`, `getSelectedItem()` und `select(int)`. Auch hier wird `countItems()` in Java-1.2-Programmen durch `getItemCount()` ersetzt.

Da mehr als nur ein Element in einem Listenfeld ausgewählt werden kann, stehen noch die beiden folgenden Methoden zur Verfügung:

- Die Methode `getSelectedIndexes()` gibt ein Integer-Array zurück, das die Indexpositionen aller ausgewählten Einträge enthält.
- Die Methode `getSelectedItems()` gibt ein String-Array zurück, das den Text aller ausgewählten Einträge beinhaltet.

Bildlaufleisten

Bildlaufleisten sind Komponenten, die es ermöglichen, einen Wert auszuwählen, indem Sie ein kleines Kästchen zwischen zwei Pfeilen bewegen. Viele Komponenten haben integrierte Bildlaufleisten, darunter mehrzeilige Textfelder und Listenfelder. Listenfelder werden mit der Klasse `Scrollbar` erzeugt. Eine Bildlaufleiste kann entweder horizontal oder vertikal sein.

Bei der Erzeugung von Bildlaufleisten werden normalerweise der Minimal- und der Maximalwert des Wertebereiches, aus dem mit dieser Komponente Werte gewählt werden können, festgelegt.

Um eine Bildlaufleiste, zu erstellen, können Sie die folgenden Konstruktoren verwenden:

- `Scrollbar()` erzeugt eine vertikale Bildlaufleiste, deren Minimal- und Maximalwert 0 ist.
- `Scrollbar(int)` erzeugt eine Bildlaufleiste, deren Minimal- und Maximalwert 0 ist, mit der angegebenen Orientierung. Über Klassenvariablen wird die Orientierung der Bildlaufleiste festgelegt. Die entsprechende Klassenvariable wird dem Konstruktor als Argument übergeben: `Scrollbar.HORIZONTAL` für eine horizontale Bildlaufleiste und `Scrollbar.VERTICAL` für eine vertikale.

Sie können auch den dritten Konstruktor verwenden, der fünf Integer-Argumente erwartet: `Scrollbar(int, int, int, int, int)`. Im folgenden werden die einzelnen Argumente der Reihe nach beschrieben:

- Die Orientierung der Bildlaufleiste (entweder `Scrollbar.HORIZONTAL` oder `Scrollbar.VERTICAL`).

- Der Anfangswert der Bildlaufleiste. Dieser sollte entweder gleich dem Minimal- bzw. dem Maximalwert sein oder zwischen diesen Werten liegen.
- Die gesamte Breite oder Höhe des Kästchens, das zur Auswahl eines Wertes verwendet wird. Dieser Wert kann auch 0 sein, wenn die Standardgröße verwendet wird.
- Der Minimalwert der Bildlaufleiste.
- Der Maximalwert der Bildlaufleiste.

Listing 11.10 zeigt ein einfaches Applet, das eine Bildlaufleiste anzeigt. Das GridLayout -Objekt wird zusammen mit der setLayout()-Methode des Applets verwendet, um ein Layout zu ermöglichen, in dem eine Bildlaufleiste ihren Container komplett ausfüllt. Über Layout-Manager lernen Sie morgen mehr.

Listing 11.10: Der komplette Quelltext von Slider.java

```

1: import java.awt.*;
2:
3: public class Slider extends java.applet.Applet {
4:     GridLayout gl = new GridLayout(1,1);
5:     Scrollbar bar = new Scrollbar(Scrollbar.HORIZONTAL,
6:     50,0,1,100);
7:
8:     public void init() {
9:         setLayout(gl);
10:        add(bar);
11:    }
12: }
```

Unabhängig davon, welche Werte für die Breite und die Höhe des Applet-Fensters verwendet werden, füllt die Bildlaufleiste die gesamte Fläche aus. Abbildung 11.12 wurde mit dem folgenden Tag erzeugt:

```
<APPLET CODE="Slider.class" WIDTH=500 HEIGHT=20>
</APPLET>
```

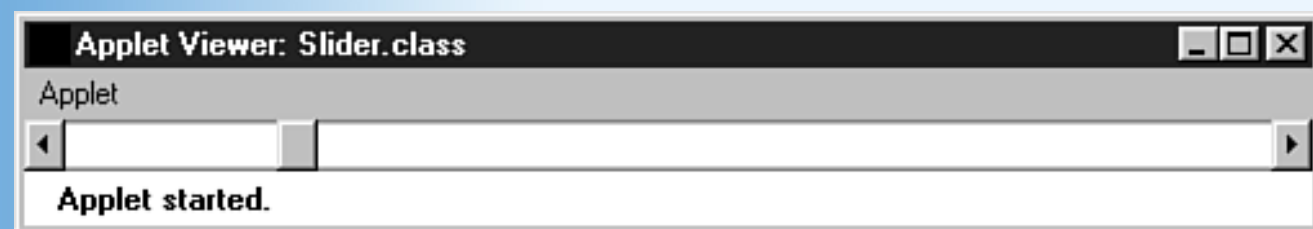


Abbildung 11.12:
Das Slider-Applet

Die Klasse Scrollbar bietet diverse Methoden, mit denen sich die Werte einer Bildlaufleiste verwalten lassen:

- Die Methode getValue() gibt den aktuellen Wert der Bildlaufleiste zurück.
- Die Methode setValue(int) legt den aktuellen Wert der Bildlaufleiste fest.

Canvas

Ein Canvas (zu deutsch Leinwand) ist eine Komponente, die im wesentlichen zur Anzeige von Bildern und Animationen in einer Benutzeroberfläche dient. Sie können auch auf andere Komponenten zeichnen, wie Sie es bereits mit dem Applet-Fenster im Laufe dieses Buches getan haben. Ein Canvas stellt aber das einfachste Objekt für diese Aufgaben dar.

Um einen Canvas zu verwenden, müssen Sie eine Subklasse der Klasse Canvas erstellen. Diese Subklasse verarbeitet in ihrer paint()-Methode alle notwendigen Ausgaben.

Sobald Sie eine Canvas-Subklasse erzeugt haben, kann diese in einem Programm verwendet werden. Dazu rufen Sie deren Konstruktor auf und fügen das neue Canvas-Objekt in einen Container ein.

Dies wird in dem Applet Crosshair demonstriert (siehe Listing 11.11). Dieses Applet zeichnet ein Zielkreuz in die Mitte des Applet-Fensters. Außerdem verschiebt es den Mittelpunkt des Zielkreuzes bei einer Größenänderung des Fensters entsprechend.

Listing 11.11: Der komplette Quelltext von Crosshair.java

```

1: import java.awt.*;
2:
3: public class Crosshair extends java.applet.Applet {
4:     GridLayout gl = new GridLayout(1,1);
5:     MyCanvas can = new MyCanvas();
6:
7:     public void init() {
8:         setLayout(gl);
9:         add(can);
10:    }
11:
12: }
13:
14: class MyCanvas extends java.awt.Canvas {
15:     public void paint(Graphics g) {
16:         int x = size().width / 2;
17:         int y = size().height / 2;
18:         g.setColor(Color.black);
19:         g.drawLine(x-10,y,x-2,y);
20:         g.drawLine(x+10,y,x+2,y);
21:         g.drawLine(x,y-10,x,y-2);
22:         g.drawLine(x,y+10,x,y+2);
23:     }
24: }
```

Das Applet kann mit einem Fenster beliebiger Größe getestet werden. Das folgende <APPLET>-Tag wurde verwendet, um die Ausgabe in Abbildung 11.13 zu erzeugen:

```
<APPLET CODE="Crosshair.class" WIDTH=100 HEIGHT=100>
</APPLET>
```

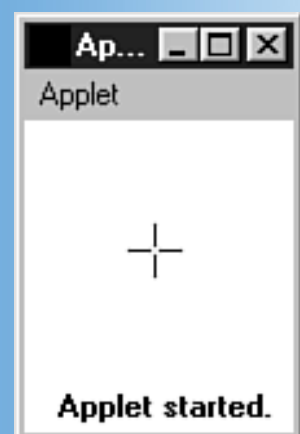


Abbildung 11.13: Das Crosshair-Applet

In Listing 11.11 befinden sich zwei Klassen. Die erste, Crosshair, stellt das Applet selbst dar. Die zweite, in den Zeilen 14-24, ist die Klasse MyCanvas, die eine Subklasse der Klasse Canvas ist.

In der Klasse Crosshair passiert folgendes:

- In Zeile 4 wird ein GridLayout-Objekt erzeugt, das in Zeile 6 als Layout-Manager für die Klasse festgelegt wird.
- In Zeile 5 wird ein MyCanvas-Objekt mit dem Namen can erzeugt. Dieses verwendet die Canvas-Subklasse, die in den Zeilen 14-24 erstellt wird.
- In der Zeile 9 wird can in das Applet-Fenster eingefügt. Da ein GridLayout-Layout-Manager verwendet wird, füllt der Canvas die gesamte Fläche.

Die meiste Arbeit wird in diesem Projekt von der Hilfsklasse myCanvas erledigt. Hier passiert folgendes:

- Die Zeilen 16 und 17 ermitteln den Mittelpunkt des Applet-Fensters. Dies wird dynamisch bei jedem Neuzeichnen des Canvas erledigt. Die Variablen `size().width` und `size().height` beinhalten die Breite und die Höhe des Canvas. Durch die Division durch 2 ergibt sich der Mittelpunkt. Wenn Sie kein 1.02-Applet schreiben, sollten Sie die Variablen `getSize().width` und `getSize().height` verwenden, um bei der Kompilierung des Programms Warnungen wegen nicht mehr verwendeter Elemente zu vermeiden.
- In Zeile 18 wird Schwarz als Hintergrundfarbe des Canvas gesetzt. Beachten Sie bitte, daß sich dieser Methodenaufruf an das Graphics-Objekt richtet und nicht an den Canvas selbst. Das Graphics-Objekt, das an die `paint()`-Methode übergeben wird, verarbeitet alle Zeichenoperationen, die auf das Objekt angewendet werden.
- Die Zeilen 19-22 verwenden x,y-Koordinaten des Mittelpunktes, um vier Linien im Stil eines Fadenkreuzes um diesen herum zu zeichnen. Jede Linie ist acht Pixel lang und endet zwei Pixel vor dem Mittelpunkt.

Zusammenfassung

Sie wissen nun, wie Sie eine Benutzerschnittstelle im Fenster eines Applets mit der Standardpalette der Sprache - den Komponenten des Abstract Windowing Toolkit - malen.

Dieses Toolkit beinhaltet Klassen für viele der Elemente, wie Schaltflächen, Bildlaufleisten, Listenfelder usw., die Sie in einem Programm erwarten. Diese Komponenten verwenden Sie, indem Sie eine Instanz von deren Klasse erzeugen und diese einem Container wie dem Applet-Fenster mit dessen `add()`-Methode hinzufügen.

Heute haben Sie einiges darüber gelernt, wie Sie Funktionalität schaffen, indem Sie Komponenten entwickeln und diese einem Programm hinzufügen.

Im Laufe der nächsten zwei Tage lernen Sie zwei weitere Dinge, die nötig sind, damit eine Benutzerschnittstelle verwendbar wird:

- Die Form: So arrangieren Sie die Komponenten, damit diese eine komplette Benutzerschnittstelle bilden.
- Feedback: So erhalten Sie über diese Komponenten Eingaben von einem Benutzer.

Fragen und Antworten

Frage:

Warum sollte ich bei all diesen verworfenen Methoden, die sich im Abstract Windowing Toolkit von Java 1.2 befinden, überhaupt noch Applets in Java 1.02 schreiben?

Antwort:

Im Idealfall sollten Sie nichts über frühere Versionen von Java lernen müssen, wenn Sie sich mit Java 1.2 beschäftigen. Allerdings sind die führenden Browser-Hersteller sehr langsam damit, die Unterstützung von Versionen der Sprache nach 1.02 einzuführen. Und wie es momentan beim Schreiben dieses Buches scheint, wird Microsoft Java 1.1 nie voll unterstützen, geschweige denn Java 1.2. Aus diesem Grund bleibt Java 1.02 der Standard bei der Programmierung von Applets. Sun arbeitet an einer Möglichkeit für Applet-Entwickler, deren eigene Laufzeitumgebung für ein Applet festzulegen. Dadurch würde es möglich werden, Java-1.2-Applets zu schreiben und dabei sicher zu sein, daß die Anwender mit einem Java-fähigen Browser dieses Applet auch ausführen können.

Frage:

Meine Java-Entwicklungsumgebung erlaubt es, die Benutzerschnittstelle eines Programms visuell zu erstellen - ich kann Schaltflächen und andere Komponenten per Drag&Drop einfügen und mit der Maus arrangieren. Muß ich den Umgang mit dem Abstract Windowing Toolkit trotzdem erlernen?

Antwort:

Wenn Sie mit den Ergebnissen, die Sie erhalten, zufrieden sind und in Ihre Fähigkeit vertrauen, diese Benutzerschnittstelle in einem funktionierenden Programm zu verwenden, dann ist das Abstract Windowing Toolkit keine Notwendigkeit. Allerdings ist es eines der Hauptprojekte dieses Buches, eine funktionierende grafische Benutzerschnittstelle mit dem AWT zu erstellen. Sie erhalten dadurch Fähigkeiten, die Ihnen in anderen Bereichen von Java nützen.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).

Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Tag 12

Benutzerschnittstellen entwerfen

Der Entwurf einer grafischen Benutzerschnittstelle ist mit dem Malen vergleichbar. Allerdings können Sie momentan nur einer Kunstrichtung nachgehen: dem abstrakten Expressionismus. Sie können Komponenten in eine Benutzerschnittstelle einfügen, allerdings haben Sie nur wenig Kontrolle darüber, wo diese plaziert werden.

Um eine gewisse Form in die Schnittstelle, die Sie mit dem *Abstract Windowing Toolkit* entwerfen, zu bekommen, müssen Sie eine Reihe von Klassen verwenden, die als Layout-Manager bezeichnet werden.

Heute lernen Sie fünf verschiedene Layout-Manager zu verwenden, um Komponenten auf einer Benutzerschnittstelle anzuordnen. Sie werden die Vorteile der Flexibilität des Abstract Windowing Toolkit von Java nutzen. Dies wurde so entworfen, daß sich die Ergebnisse auf all den verschiedenen Plattformen, die die Sprache unterstützen, präsentiert werden können.

Für den Fall, daß ein Arrangement nicht das erfüllt, was Sie sich für ein Programm vorgestellt haben, lernen Sie, wie Sie verschiedene Layout-Manager auf derselben Schnittstelle zugleich einsetzen.

Lassen Sie uns mit den elementaren Layout-Managern beginnen.

Das elementare Layout einer Benutzerschnittstelle

Wie Sie gestern gelernt haben, ist eine grafische Benutzerschnittstelle, die mit dem AWT entworfen wurde, stets in Fluß. Die Veränderung der Größe eines Fensters kann eine nachhaltige Auswirkung auf Ihre Benutzerschnittstelle haben, da Komponenten an Stellen verschoben werden können, die nicht dem entsprechen, was Sie sich dafür vorgestellt haben.

Dieser Fluß ergibt sich aus einer Notwendigkeit: Java ist auf vielen verschiedenen Plattformen implementiert und zwischen diesen gibt es feine Unterschiede in der Art, wie diese z.B. Schaltflächen, Bildlaufleisten usw. anzeigen.

Bei Programmiersprachen, wie z.B. Microsoft Visual Basic, wird die Position einer Komponente in einem Fenster genau über deren x,y-Koordinaten festgelegt. Einige Java-Entwicklungstools bieten eine ähnliche Kontrolle über eine Benutzerschnittstelle. Dazu verwenden diese spezielle Klassen.

Bei der Verwendung des Abstract Windowing Toolkit erhält ein Programmierer mit Layout-Managern mehr Kontrolle über das Layout der Benutzerschnittstelle.

Das Layout einer Benutzerschnittstelle

Ein Layout-Manager legt fest, wie Komponenten arrangiert werden, sobald diese einem Container hinzugefügt werden.

Der Standard-Layout-Manager ist die Klasse `FlowLayout`. Diese Klasse läßt Komponenten in der Reihenfolge, in der diese in einen Container eingefügt wurden, von links nach rechts fließen. Sobald kein Platz mehr zur Verfügung steht, wird eine neue Zeile mit Komponenten direkt unter dieser Zeile angefangen - wieder von links nach rechts.

Das AWT beinhaltet fünf elementare Layout-Manager: `FlowLayout`, `GridLayout`, `BorderLayout`, `CardLayout` und `GridBagLayout`. Um einen Layout-Manager für einen Container zu erzeugen, wird eine Instanz des Layout-Managers erstellt. Dazu ist eine Anweisung wie die folgende notwendig:

```
FlowLayout flo = new FlowLayout();
```

Nachdem Sie einen Layout-Manager erzeugt haben, weisen Sie ihn einem Container über dessen `setLayout()`-Methode zu. Der Layout-Manager muß festgelegt worden sein, bevor Komponenten dem Container hinzugefügt werden. Wenn kein Layout-Manager festgelegt wurde, wird `FlowLayout` verwendet.

Die folgenden Anweisungen stellen den Ausgangspunkt für ein Applet dar, das einen Layout-Manager erzeugt und die Methode `setLayout()` verwendet, so daß dieser Layout-Manager die Anordnung aller Komponenten, die dem Applet-Fenster hinzugefügt werden, kontrolliert:

```
public class Starter extends java.applet.Applet {
    FlowLayout lm = new FlowLayout();

    public void init() {
        setLayout(lm);
    }
}
```

Nachdem der Layout-Manager festgelegt wurde, können Sie damit beginnen, Komponenten in den Container, der von diesem Layout-Manager kontrolliert wird, einzufügen. Bei manchen Layout-Managern, wie z.B. `FlowLayout`, spielt die Reihenfolge, in der die Komponenten hinzugefügt werden, eine wesentliche Rolle. In den einzelnen Abschnitten des heutigen Tages, wenn Sie mit den einzelnen Layout-Managern arbeiten, lernen Sie mehr dazu.

Die Klasse `FlowLayout`

Die `FlowLayout`-Klasse stellt das einfachste Layout dar. Mit diesem Layout werden die Komponenten nacheinander zeilenweise in das Panel eingefügt. Paßt eine Komponente nicht in eine Zeile, wird sie automatisch auf die nächste Zeile umbrochen. Das `FlowLayout` hat eine Ausrichtung, die die Ausrichtung aller Zeilen vorgibt. Standardmäßig ist jede Zeile zentriert.



`FlowLayout` ordnet Komponenten zeilenweise von links nach rechts an. Die Zeilen werden entweder nach links, rechts oder zentriert ausgerichtet.

Um ein einfaches `FlowLayout` mit zentrierter Ausrichtung zu erstellen, verwenden Sie in der Panel-Initialisierung folgende Codezeile (da dies das Panel-Standardlayout ist, können Sie, wenn Sie wollen, diese Zeile auslassen):

```
setLayout(new FlowLayout());
```

Das Applet in Listing 12.1 zeigt sechs Schaltflächen an, die mit dem `FlowLayout` angeordnet werden. Da dem `FlowLayout`-Konstruktor die Klassenvariable `FlowLayout.LEFT` übergeben wird, werden die einzelnen Komponenten an der linken Seite des Applets ausgerichtet.

Listing 12.1: Der gesamte Quelltext von `Alphabet.java`

```
1: import java.awt.*;
2:
3: public class Alphabet extends java.applet.Applet {
4:     Button a = new Button("Alibi");
5:     Button b = new Button("Burglar");
6:     Button c = new Button("Corpse");
7:     Button d = new Button("Deadbeat");
8:     Button e = new Button("Evidence");
9:     Button f = new Button("Fugitive");
10:    FlowLayout lm = new FlowLayout(FlowLayout.LEFT);
11:
12:    public void init() {
13:        setLayout(lm);
14:        add(a);
15:        add(b);
16:        add(c);
17:        add(d);
18:        add(e);
```

```

19:         add(f);
20:     }
21: }

```

Das folgende <APPLET>-Tag wurde verwendet, um das Applet im Appletviewer, wie in Abbildung 12.1 dargestellt, anzuzeigen.

```

<applet code="Alphabet.class" height=120 width=220>
</applet>

```

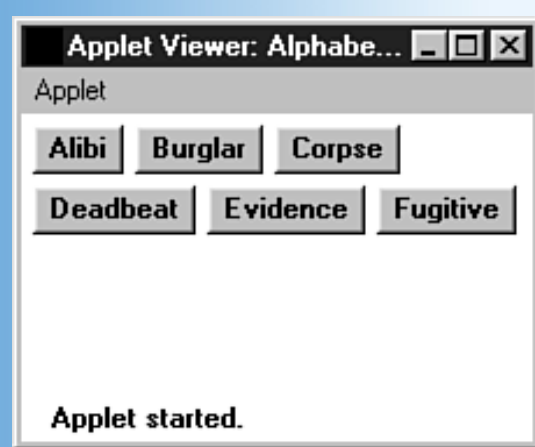


Abbildung 12.1:
Sechs mit einem FlowLayout-Manager angeordnete Schaltflächen

Um ein FlowLayout mit einer rechten oder linken Ausrichtung zu erstellen, fügen Sie die Klassenvariable FlowLayout.RIGHT oder FlowLayout.LEFT als Argument ein:

```
setLayout(new FlowLayout(FlowLayout.LEFT));
```

Mit Hilfe des FlowLayout können Sie auch horizontale und vertikale Abstandswerte setzen. Der Abstand ist die Zahl der Pixel zwischen Komponenten in einem Panel. Standardmäßig sind die horizontalen und vertikalen Abstandswerte drei Pixel, was sehr eng ist. Der horizontale Abstand ist links und rechts zwischen Komponenten, der vertikale oben und unten. Um den Abstand zu erhöhen, fügen Sie in den FlowLayout- Konstruktor ganzzahlige Argumente ein. Sie können einen Abstand von 30 Punkten in horizontaler und 10 in vertikaler Ausrichtung wie folgt einfügen:

```
setLayout(new FlowLayout(FlowLayout.LEFT, 30, 10));
```

Die Klasse GridLayout

Layouts mit GridLayout bieten mehr Kontrolle über die Anordnung von Komponenten in einem Panel. Mit einem GridLayout können Sie den Panel-Bereich in Zeilen und Spalten aufteilen. Jede Komponente, die Sie dann in das Panel einfügen, wird in einer Zelle des Rasters von links oben nach rechts unten in das Raster eingefügt (hier ist die Reihenfolge, in der die add()-Methode aufgerufen wird, für das Bildschirmlayout relevant).

Um ein GridLayout zu erstellen, geben Sie die Anzahl der gewünschten Zeilen und Spalten in einer neuen Instanz der GridLayout-Klasse an. Hier ein GridLayout mit drei Zeilen und zwei Spalten. In das Raster werden sechs Schaltflächen eingefügt.

```
import java.awt.*;
```

```
public class GridLayoutTest extends java.applet.Applet {
```

```

    public void init() {
        setLayout(new GridLayout(3,2);
        add(new Button("One"));
        add(new Button("Two"));
        add(new Button("Three"));
        add(new Button("Four"));
        add(new Button("Five"));
    }
}

```

```

        add(new Button("Six"));
    }
}

```

Auch bei einem GridLayout können Sie den horizontalen und vertikalen Abstand zwischen den Komponenten bestimmen. Hierfür fügen Sie die entsprechenden Pixel- Werte ein:

```
setLayout(new GridLayout(3, 3, 10, 30));
```

Der Standardabstand zwischen einzelnen Komponenten beträgt sowohl horizontal als auch vertikal null Pixel.

Listing 12.2 beinhaltet ein Applet, das ein Raster mit 3 Zeilen, 3 Spalten und einem vertikalen und horizontalen Abstand von 10 Pixeln zwischen den einzelnen Komponenten erzeugt.

Listing 12.2: Der gesamte Quelltext von Bunch.java

```

1: import java.awt.*;
2:
3: public class Bunch extends java.applet.Applet {
4:     GridLayout family = new GridLayout(3,3,10,10);
5:     Button marcia = new Button("Marcia");
6:     Button carol = new Button("Carol");
7:     Button greg = new Button("Greg");
8:     Button jan = new Button("Jan");
9:     Button alice = new Button("Alice");
10:    Button peter = new Button("Peter");
11:    Button cindy = new Button("Cindy");
12:    Button mike = new Button("Mike");
13:    Button bobby = new Button("Bobby");
14:
15:    public void init() {
16:        setLayout(family);
17:        add(marcia);
18:        add(carol);
19:        add(greg);
20:        add(jan);
21:        add(alice);
22:        add(peter);
23:        add(cindy);
24:        add(mike);
25:        add(bobby);
26:    }
27: }

```

Abbildung 12.2 zeigt dieses Applet auf einer Seite mit dem folgenden <APPLET>-Tag:

```

<applet code="Bunch.class" height=160 width=160>
</applet>

```



Abbildung 12.2:
Neun Schaltflächen, die in einem 3× 3-Raster angeordnet wurden

Beachten Sie bei den Schaltflächen in Abbildung 12.2 bitte, daß diese so vergrößert werden, daß sie den gesamten verfügbaren Raum in der jeweiligen Zelle einnehmen. Dies ist ein wesentlicher Unterschied zwischen dem GridLayout und anderen Layout- Managern. Beim GridLayout nehmen die einzelnen Komponenten immer den gesamten Raum einer Zelle ein. Wenn Sie das Bunch-Applet mit dem Appletviewer laden, dann werden Sie feststellen, daß sich die Größe der Schaltflächen ändert, sobald Sie die Größe des Applet-Fensters ändern.

Die Klasse BorderLayout

Layouts mit BorderLayout verhalten sich anders als FlowLayout und GridLayout. Wenn Sie eine Komponente in ein Panel einfügen, das auf einem BorderLayout basiert, müssen Sie seine Anordnung als geographische Richtung angeben: Nord, Süd, Ost, West oder Mitte. (Siehe Abbildung 12.3.) Die Komponenten rund um die Kanten werden in der benötigten Größe ausgelegt. Falls es eine Komponente in der Mitte gibt, erhält sie den restlichen Platz zugeteilt.

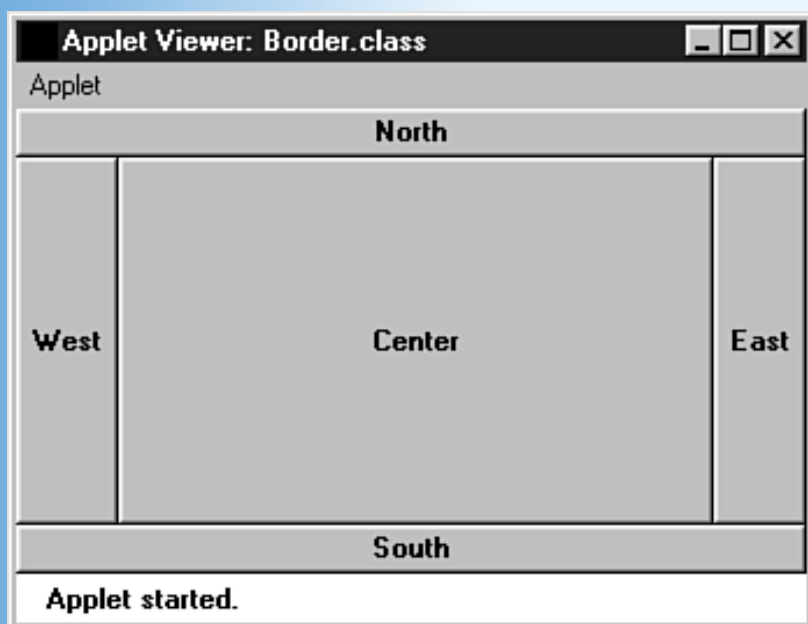


Abbildung 12.3:
Anordnung der Komponenten in einem BorderLayout

Um ein BorderLayout zu erstellen, verfahren Sie wie bei den anderen Layouts; fügen Sie dann die einzelnen Komponenten mit einer speziellen, zwei Argumente beinhaltenden add()-Methode ein. Das erste Argument ist eine Zeichenkette, die die Position der Komponente im Layout bezeichnet, und das zweite die einzufügende Komponente:

```
add("North", new TextField("Title", 50));
```

Sie können diese Form von `add()` auch für andere Layout-Manager verwenden; das Argument für die Zeichenkette wird einfach ignoriert, falls es nicht benötigt wird.

Mit dem Code aus Listing 12.3 erstellen Sie das in Abbildung 12.3 gezeigte BorderLayout :

Listing 12.3: Der gesamte Quelltext von `Border.java`

```

1: import java.awt.*;
2:
3: public class Border extends java.applet.Applet {
4:     BorderLayout b = new BorderLayout();
5:     Button north = new Button("North");
6:     Button south = new Button("South");
7:     Button east = new Button("East");
8:     Button west = new Button("West");
9:     Button center = new Button("Center");
10:
11:     public void init() {
12:         setLayout(b);
13:         add("North", north);
14:         add("South", south);
15:         add("East", east);
16:         add("West", west);
17:         add("Center", center);
18:     }
19: }
```

Zusätzlich wurde das folgende `<APPLET>`-Tag verwendet:

```

<applet code="Border.class" height=120 width=120>
</applet>
```

Auch bei BorderLayout sind horizontale und vertikale Abstände möglich. Beachten Sie, daß sich die Nord- und Südkomponenten über die gesamte Kante des Panels erstrecken, so daß der Abstand auf Kosten des Platzes für die Komponenten in Ost, West und Mitte entsteht. Um Abstände in ein BorderLayout einzufügen, tragen Sie, wie bei den anderen Layout-Managern, die Pixel-Werte in den Konstruktor ein:

```
setLayout(new BorderLayout(10, 10));
```

Verschiedene Layout-Manager gleichzeitig

An diesem Punkt werden Sie sich vielleicht fragen, wie Sie mit den Layout-Managern des Abstract Windowing Toolkit die Benutzerschnittstelle, die Sie sich vorgestellt haben, entwerfen sollen. Die einzelnen Layout-Manager für sich genommen haben wahrscheinlich nicht genau die Eigenschaften, die Sie benötigen.

Um das Layout zu finden, das genau richtig ist, müssen Sie oftmals mehr als einen Layout-Manager in derselben Benutzerschnittstelle kombinieren.

Dies erreichen Sie, indem Sie in den Haupt-Container (z.B. ein Applet-Fenster) weitere Container einfügen. Jedem dieser Unter-Container weisen Sie einen eigenen Layout-Manager zu.

Diese Unter-Container sind Panels - Objekte der Klasse `Panel`. Panels sind Container, die zur Gruppierung von Komponenten verwendet werden. Zwei Dinge sollten Sie sich in bezug auf den Umgang mit Panels merken:

- Das Panel wird mit Komponenten gefüllt, bevor es in einen übergeordneten Container eingefügt wird.
- Das Panel verfügt über seinen eigenen Layout-Manager.

Panels werden mit einem einfachen Aufruf des Konstruktors der `Panel`-Klasse erzeugt, wie das in dem folgenden Beispiel gezeigt wird:

```
Panel pane = new Panel();
```


Der Layout-Manager für das Panel wird über einen Aufruf der Methode `setLayout()` des Panels festgelegt. Diese funktioniert genauso, wie der Aufruf der `setLayout()`-Methode, die Sie zuvor für das Applet-Fenster verwendet haben - sowohl Applet als auch Panel sind Subklassen der Klasse Container, und sie erben das Verhalten für das Layout-Management von deren Superklasse.

Die folgenden Anweisungen erzeugen einen Layout-Manager und weisen diesen einem Panel-Objekt mit dem Namen `pane` zu:

```
BorderLayout bo = new BorderLayout();
pane.setLayout(bo);
```

Komponenten werden dem Panel über einen Aufruf der `add()`-Methode hinzugefügt. Dies funktioniert hier genauso wie bei anderen Containern, wie z.B. Applets.

Die folgende Anweisung fügt einen Textbereich mit dem Namen `dialogue` in das Panel -Objekt `pane` ein:

```
pane.add(dialogue);
```

Sie werden in den weiteren Beispielprogrammen des heutigen Tages noch viele Beispiele für die Verwendung von Panels sehen.

Fortgeschrittene Layout-Manager

Zusätzlich zu den drei Layout-Managern, die Sie bereits kennengelernt haben - `FlowLayout`, `GridLayout` und `BorderLayout` -, beinhaltet das Abstract Windowing Toolkit zwei weitere, ausgefeiltere Layout-Manager. Die beiden Layout-Manager `CardLayout` und `GridBagLayout` können auch mit anderen Layout-Managern zusammen verwendet werden, indem man Container ineinander verschachtelt.

Die Klasse CardLayout

Layouts mit `CardLayout` unterscheiden sich von anderen Layouts. Wenn Sie in einen der anderen Layout-Manager Komponenten einfügen, erscheinen diese Komponenten sofort am Bildschirm. `CardLayout` wird dazu verwendet, um eine Art Diaschau der Komponenten zu erzeugen. Falls Sie je mit HyperCard auf dem Macintosh gearbeitet haben, kennen Sie das Prinzip von `CardLayout`.

Die Komponenten, die Sie beim Erstellen eines `CardLayout` in das äußere Panel einfügen, werden im allgemeinen als weitere Container-Komponenten - normalerweise sind es Panels - behandelt. Sie können dann für die einzelnen Karten je ein anderes Layout verwenden, so daß jeder Bildschirm anders aussieht.



Cards (Karten) in einem `CardLayout` sind unterschiedliche Panels, die nacheinander eingefügt und angezeigt werden. Wenn Sie sich ein Kartenspiel vorstellen, werden Sie verstehen, was gemeint ist; es kann jeweils nur eine Karte angezeigt werden, aber Sie können zwischen den Karten wechseln.

Jede Karte, die Sie in das Panel einfügen, wird benannt. Um jeweils zwischen den Container-Karten zu blättern, können Sie in der `CardLayout`-Klasse definierte Methoden verwenden, mit denen Sie zu einer benannten Karte gelangen, vorwärts oder rückwärts, oder zur ersten oder letzten Karte blättern. Natürlich steht Ihnen zum Aufrufen dieser Methoden eine Gruppe von Schaltflächen zur Verfügung, die Ihnen das Navigieren durch das `CardLayout` erleichtern.

Mit den folgenden zwei einfachen Anweisungen wird ein `CardLayout` mit drei Karten erstellt:

```
setLayout(new CardLayout());
//Karten hinzufügen
Panel one = new Panel()
add("first", one);
Panel two = new Panel()
add("second", two);
```

```
Panel three = new Panel()
add("third", three);
```

```
//navigieren
show(this, "second"); //Sprung zu Karte namens "second"
show(this, "third");  //Sprung zu Karte namens "third"
previous(this);       //Zurück zur zweiten Karte
first(this);          //Sprung zur ersten Karte
```

In einem Programm, das das CardLayout verwendet, wird der Wechsel zwischen den Karten normalerweise durch eine Aktion des Benutzers ausgelöst. In einem Programm, das Adressen auf verschiedenen Karten anzeigt, könnte der Benutzer eine Karte für die Anzeige auswählen, indem er einen Eintrag in einer Liste selektiert. Als Alternative dazu verwendet das Applet in Listing 12.4 eine Animation in einem Thread, um zwischen den einzelnen Panels in dem CardLayout umzuschalten.

Listing 12.4: Der gesamte Quelltext von BurmaShave.java

```
1: import java.awt.*;
2:
3: public class BurmaShave extends java.applet.Applet
4:     implements Runnable {
5:
6:     CardLayout card = new CardLayout();
7:     Label[] lab = new Label[6];
8:     int current = 0;
9:     Thread runner;
10:
11:     public void start() {
12:         if (runner == null) {
13:             runner = new Thread(this);
14:             runner.start();
15:         }
16:     }
17:
18:     public void stop() {
19:         runner = null;
20:     }
21:
22:     public void init() {
23:         lab[0] = new Label("Grandpa's beard");
24:         lab[1] = new Label("Was stiff and coarse.");
25:         lab[2] = new Label("And that's what caused");
26:         lab[3] = new Label("His fifth");
27:         lab[4] = new Label("Divorce.");
28:         lab[5] = new Label("Burma Shave.");
29:         setLayout(card);
30:         for (int i = 0; i < 6; i++)
31:             add("Card " + i, lab[i]);
32:     }
33:
34:     public void run() {
35:         Thread thisThread = Thread.currentThread();
36:         while (runner == thisThread) {
37:             card.show(this, "Card " + current);
38:             current++;
39:             if (current > 5)
40:                 current = 0;
41:             repaint();
```

```

42:         try {
43:             Thread.sleep(5000);
44:         } catch (InterruptedException e) { }
45:     }
46: }
47: }

```

Das folgende <APPLET>-Tag wurde verwendet, um die Ausgabe in Abbildung 12.4 zu erzeugen.

```

<applet code="BurmaShave.class" height=80 width=160>
</applet>

```



Abbildung 12.4:
Eine Karte aus einem CardLayout mit mehreren Karten

Das BurmaShave-Applet verwendet ein CardLayout mit sechs Karten. Jede Karte ist eine Label-Komponente. Die Animation wird über den Wechsel zwischen den einzelnen Karten erzeugt.

Einige Anmerkungen zu dem Applet:

- Zeile 7 - Das Array lab wird erzeugt, um die sechs Labels aufzunehmen.
- Zeile 8 - Die Variable current wird deklariert und initialisiert. Diese Variable verfolgt die Nummer der aktuellen Karte.
- Zeilen 23 bis 28 - Die sechs Label-Objekte werden erzeugt. Jedem wird ein Teil des Textes eines Werbeslogans zugewiesen.
- Zeile 29 - CardLayout wird als Layout-Manager für das Applet festgelegt.
- Zeilen 30 und 31 - Mit einer for-Schleife werden die sechs Labels in dem lab- Array dem Applet-Fenster als Karten des CardLayouts hinzugefügt. Jeder Karte wird ein Name zugewiesen, der mit dem Wort »Card« beginnt. Diesem folgt ein Leerzeichen und anschließend wird eine Nummer von 0 bis 5 hinzugefügt (z.B. »Card 5«).
- Zeile 37 - Die Methode show() der Klasse CardLayout wird aufgerufen, um die aktuelle Karte anzuzeigen. Der Name der Karte setzt sich aus dem Wort »Card«, einem Leerzeichen und dem Wert der Variablen current zusammen.
- Zeile 38 - Der Wert der Variablen current wird um 1 erhöht.
- Zeile 39 und 40 - Die Variable current wird auf 0 gesetzt, falls deren Wert höher als 5 ist.

Die Klasse GridBagLayout

Das GridBagLayout haben wir uns bis zum Schluß aufgehoben. Es ist zwar das stärkste Instrument in der Verwaltung von AWT-Layouts, jedoch ist es gleichzeitig auch äußerst kompliziert.

Genau das gewünschte Layout zu erhalten, kann sich bei Verwendung einer der vier anderen Layout-Manager manchmal recht schwierig gestalten, ohne viele Panels ineinander verschachteln zu müssen. Das GridLayout bietet eine allgemeineren Zwecken dienliche Lösung. Wie das GridLayout ermöglicht Ihnen das GridBagLayout die Anordnung Ihrer Komponenten in einem rasterähnlichen Layout. Allerdings bietet Ihnen das GridBagLayout zusätzlich die Möglichkeit, die Weite einzelner Zellen im Raster, die Proportionen zwischen Zeilen und Spalten sowie die Anordnung von Komponenten innerhalb der Zellen im Raster zu kontrollieren.

Zur Erstellung eines GridBagLayout benutzen Sie zwei Klassen: GridBagLayout, die den gesamten Layout-Manager bereitstellt, und die Klasse GridBagConstraints, die die Eigenschaften jeder Komponente im Raster bestimmt - seine Anordnung, Maße, Ausrichtung usw. Das Verhältnis zwischen GridBagLayout, GridBagConstraints und jeder Komponente bestimmt das gesamte Layout.

Zur Erstellung eines GridBagLayout in seiner einfachsten Form gehen Sie folgendermaßen vor:

1. Erstellen Sie ein GridBagLayout-Objekt, und definieren Sie es in gleicher Weise wie andere Layout-Manager als den derzeitigen Layout-Manager.
2. Erstellen Sie eine neue Instanz von GridBagConstraints.
3. Richten Sie die Rahmenbedingungen für eine Komponente ein.
4. Informieren Sie den Layout-Manager über die Komponente und Ihre GridBagConstraints .
5. Fügen Sie die Komponente in das Panel ein.

Der folgende einfache Code erstellt das Layout und erzeugt dann GridBagConstraints für eine einzelne Schaltfläche. (Machen Sie sich über die verschiedenen Werte der GridBagConstraints keine Sorgen; darüber reden wir später in diesem Abschnitt.)

```
// Layout einrichten
GridBagLayout gridbag = new GridBagLayout();
GridBagConstraints constraints = new GridBagConstraints();
setLayout(gridbag);

// Constraints für die Schaltfläche definieren
Button b = new Button("Save");
constraints.gridx = 0;
constraints.gridy = 0;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.weightx = 30;
constraints.weighty = 30;
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.CENTER;

// Constraints zu Layout und Button hinzufügen
gridbag.setConstraints(b, constraints);
add(b);
```

Wie Sie anhand des Beispiels sehen können, müssen Sie alle GridBagConstraints für jede Komponente, die Sie in das Panel einfügen wollen, setzen. Hinzu kommt, daß GridBagConstraints wirklich nicht so einfach sind; sie haben viele verschiedene Werte, von denen viele zueinander in Beziehung stehen, was bedeutet, daß die Änderung eines Werts direkte Auswirkungen auf andere haben kann. Bei den zahlreichen GridBagConstraints ist es hilfreich, planmäßig vorzugehen und sich mit jeder GridBagConstraints -Eigenschaft einzeln zu beschäftigen.

Erster Schritt: Rasterentwurf

Der erste Schritt zur Erstellung eines GridBagLayout besteht in einem Entwurf auf Papier. Die Erstellung des Designs Ihrer Benutzeroberfläche in Form einer Skizze vorab - noch bevor Sie auch nur eine Codezeile schreiben - wird auf lange Sicht gesehen enorm hilfreich sein, wenn Sie versuchen herauszufinden, wo was hinkommt. Wenden Sie sich also einmal von Ihrem Editor ab, nehmen Sie »Papier und Bleistift«, und entwerfen Sie das Raster.

Abbildung 12.5 zeigt das Panel-Layout, das Sie in diesem Beispiel konstruieren. Abbildung 12.6 zeigt dasselbe Layout mit einem aufgesetzten Raster. Ihr Layout wird ein ähnliches Raster haben, in dem Zeilen und Spalten einzelne Zellen bilden.

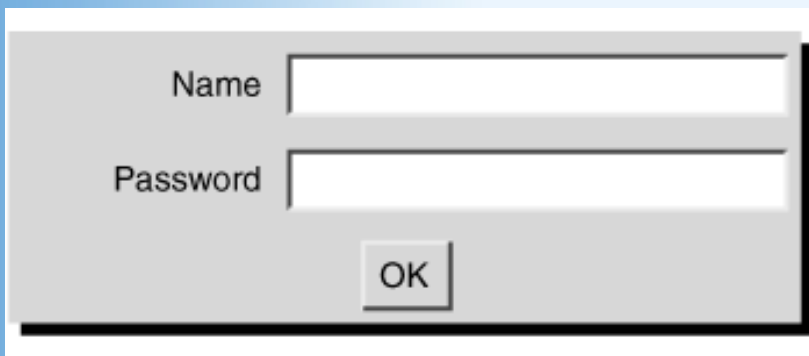


Abbildung 12.5:
Ein GridBagLayout

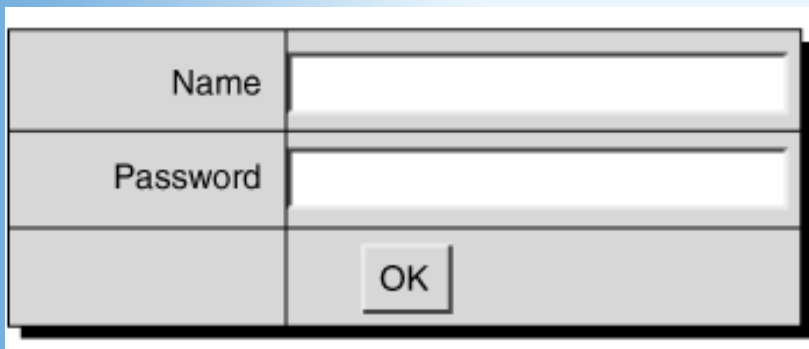


Abbildung 12.6:
Das GridBagLayout aus Abbildung 12.5 mit darüberliegendem Raster

Denken Sie beim Zeichnen Ihres Rasters daran, daß jede Komponente ihre eigene Zelle haben muß. Nur jeweils eine Komponente kann in dieselbe Zelle gesetzt werden. Umgekehrt stimmt das nicht, allerdings kann eine Komponente mehrere Zellen in x- oder y-Richtung umfassen (wie bei der Schaltfläche OK in der unteren Zeile, die zwei Spalten umfaßt). Beachten Sie, daß die Labels und Textfelder in Abbildung 12.6 ihre eigenen Raster haben und die Schaltfläche horizontal zwei Zellen umfaßt.

Setzen Sie vorerst Ihre Arbeit auf dem Papier fort, und beschriften Sie die Zellen mit ihren x- und y-Koordinaten. Sie werden später sehen, wie nützlich das ist. Es handelt sich nicht um Pixelkoordinaten, sondern um Feldkoordinaten. Die Zelle links oben ist 0,0. Die nächste Zelle rechts davon in der oberen Zeile ist 1,0. Die Zelle rechts von dieser ist 2,0. In der nächsten Zeile ist die Zelle ganz links 1,0, die nächste Zelle in dieser Zeile ist 1,1 usw. Beschriften Sie Ihre Zellen auf dem Papier mit diesen Nummern; Sie werden sie später benötigen, wenn Sie den Code für dieses Beispiel erstellen. In Abbildung 12.7 sind die Zahlen für jede Zelle des Beispiels angezeigt.

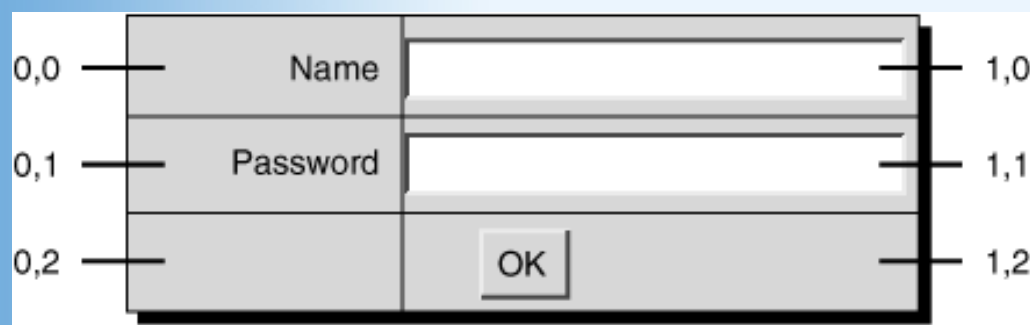


Abbildung 12.7:
Das GridBagLayout gemäß Abbildung 12.14 mit Feldkoordinaten

Zweiter Schritt: Rastererstellung in Java

Wenden Sie sich jetzt wieder Java zu, und beginnen Sie mit der Realisierung des auf dem Papier vorbereiteten Layouts. Wir konzentrieren uns zuerst ausschließlich auf das Layout - damit das Raster und die Proportionen stimmen. Dabei ist es hilfreich, nicht mit den Elementen der Benutzeroberfläche zu arbeiten. Schaltflächen dienen als

Platzhalter für die Elemente im Layout, bis alles richtig eingerichtet ist. Danach werden die Schaltflächen in richtige Elemente umgewandelt.

Um Ihr Pensum an Eigenarbeit zum Einrichten all dieser GridBagConstraints-Eigenschaften zu reduzieren, können Sie mit der Definition einer Hilfsmethode beginnen, die mehrere Werte entgegennimmt und die Rahmenbedingungen für diese Werte setzt. buildConstraints() erhält dazu sieben Argumente: ein GridBagConstraints- Objekt und sechs Ganzzahlen, die für die GridBagConstraints-Instanzvariablen gridx , gridy, gridwidth, gridheight, weightx und weighty stehen. Was deren Funktion ist, werden Sie in Kürze lernen; im Moment beschäftigen wir uns mit dem Code der Hilfsmethode, den Sie im weiteren Verlauf dieses Beispiels einsetzen:

```
void buildConstraints(GridBagConstraints gbc, int gx, int gy,
int gw, int gh, int wx, int wy) {
gbc.gridx = gx;
gbc.gridy = gy;
gbc.gridwidth = gw;
gbc.gridheight = gh;
gbc.weightx = wx;
gbc.weighty = wy;
}
```

Wenden wir uns jetzt der init()-Methode zu. In dieser wird das gesamte Layout aufgebaut. Es folgt die grundlegende Definition der Methode, in der Sie das GridBagLayout als Layout-Manager definieren und ein GridBagConstraints-Objekt erzeugen:

```
public void init() {
GridBagLayout gridbag = new GridBagLayout();
GridBagConstraints constraints = new GridBagConstraints();
setLayout(gridbag);
```

```
constraints.fill = GridBagConstraints.BOTH;
}
```

Noch ein kurzer erklärender Hinweis: Die letzte Zeile, die den Wert für constraints.fill setzt, wird später entfernt (und erklärt). Sie dient dem Zweck, daß die Komponenten die gesamte, sie beinhaltende Zelle füllen, was Ihnen die Möglichkeit gibt, zu sehen, was vor sich geht. Fügen Sie sie jetzt hinzu, und Sie werden ihren späteren Zweck verstehen.

Fügen Sie jetzt die Platzhalter-Schaltflächen in das Layout ein (denken Sie daran, daß Sie sich momentan auf einfache Rasterorganisation konzentrieren und deshalb Schaltflächen als Platzhalter für echte, später hinzuzufügende Elemente der Benutzeroberfläche verwenden). Beginnen Sie mit einer einzelnen Schaltfläche, damit Sie ein Gefühl für das Setzen der GridBagConstraints entwickeln. Dieser Code kommt in die init() -Methode direkt im Anschluß an die Zeile setLayout():

```
//Label für das Namensfeld
buildConstraints(constraints, 0, 0, 1, 1, 100, 100);
Button label1 = new Button("Name:");
gridbag.setConstraints(label1, constraints);
add(label1);
```

Diese vier Zeilen richten die GridBagConstraints für ein Objekt ein, erstellen eine neue Schaltfläche, weisen der Schaltfläche die GridBagConstraints zu und fügen sie dann in das Panel ein. Beachten Sie, daß die Rahmenbedingungen (deutsch für Constraints) für eine Komponente in einem GridBagConstraints-Objekt gespeichert werden, was bedeutet, daß die Komponente nicht einmal vorhanden sein muß, um ihre Rahmenbedingungen einzurichten.

Nun können Sie sich den Einzelheiten widmen: Welches sind die Werte für die Rahmenbedingungen, die Sie in die Hilfsmethode buildConstraints() eingebunden haben?

Die ersten beiden ganzzahligen Argumente sind die Werte gridx und gridy der Rahmenbedingungen. Sie stellen die Feldkoordinaten der Zelle dar, die diese Komponente aufnehmen soll. Erinnern Sie sich, daß Sie diese Koordinaten in Schritt 1 in Ihre Skizze geschrieben haben? Da Sie die Zellen sozusagen schon mit Nummern auf dem Papier

versehen haben, müssen Sie jetzt nur noch die richtigen Werte einsetzen. Beachten Sie bei einer mehrere Zellen umfassenden Komponente, daß die Koordinaten der Zelle sich auf die Zelle in der obersten linken Ecke beziehen.

Hier ist diese Schaltfläche in der obersten linken Ecke, und somit sind ihre Werte für `gridx` und `gridy` (die ersten beiden Argumente für `buildConstraints()`) 0 bzw. 0.

Die zweiten zwei ganzzahligen Argumente sind `gridwidth` und `gridheight`. Sie stellen keine Pixelbreiten und -höhen der Zelle dar, sondern die Anzahl der Zellen, die diese Komponente umfaßt: `gridwidth` für die Spalten und `gridheight` für die Zeilen. In unserem Beispiel umfaßt diese Komponente nur eine Zelle, und somit ist der Wert für beide 1.

Die letzten beiden ganzzahligen Argumente stehen für `weightx` und `weighty`. Sie dienen zum Einrichten der Proportionen der Zeilen und Spalten - bzw. zur Angabe von deren Breite und Tiefe. Die `Weights`-Variablen können sehr verwirrend sein, deshalb setzen Sie beide Werte jetzt einfach auf 100. Mit `Weights` werden wir uns in Schritt 3 beschäftigen.

Nachdem Sie die Rahmenbedingungen erstellt haben, können Sie sie mit der Methode `setConstraints()` an ein Objekt anbinden. Diese Methode wird in `GridBagLayout` definiert und arbeitet mit zwei Argumenten: der Komponente (hier eine Schaltfläche) und den Rahmenbedingungen für diese Schaltfläche. Zum Schluß können Sie die Schaltfläche in das Panel einfügen.

Wenn Sie die Rahmenbedingungen gesetzt und einer Komponente zugeordnet haben, können Sie das Objekt `GridBagConstraints` zur Einrichtung der Rahmenbedingungen für das nächste Objekt erneut verwenden. Hierfür duplizieren Sie diese vier Zeilen für jede Komponente im Raster, wobei Sie unterschiedliche Werte für die Methode `buildConstraints()` verwenden. Um Platz zu sparen, werden Ihnen nur die `buildConstraints()` -Methoden für die letzten vier Zellen gezeigt.

Die zweite einzufügende Zelle ist die das Textfeld für den Namen beinhaltende Zelle. Die Koordinaten für diese Zelle sind 1,0 (zweite Spalte, erste Zeile); auch in diesem Fall wird nur eine Zelle eingefaßt, und die `Weights` sind (für den Moment) auch jeweils 100:

```
buildConstraints(constraints, 1, 0, 1, 1, 100, 100);
```

Die nächsten beiden Komponenten, ein Label und ein Textfeld, sind mit den beiden vorherigen beinahe identisch; den einzigen Unterschied bilden die Koordinaten der Zelle. Das Paßwort-Label ist auf 0,1 (erste Spalte, zweite Zeile) und das Paßworttextfeld auf 1,1 (zweite Spalte, zweite Zeile) gesetzt:

```
buildConstraints(constraints, 0, 1, 1, 1, 100, 100);
```

```
buildConstraints(constraints, 1, 1, 1, 1, 100, 100);
```

Zum Schluß benötigen Sie die Schaltfläche OK, welche eine zwei Zellen auf der unteren Zeile des Panels umfassende Komponente ist. In diesem Fall sind die Zellkoordinaten die Koordinaten der Zelle ganz links oben des mehrzelligen Bereichs, den die Komponente einnimmt (0,2). Im Gegensatz zu den vorherigen Komponenten setzen Sie hier `gridwidth` auf einen von 1 verschiedenen Wert, da diese Zelle mehrere Spalten umfassen kann. `gridweight` ist 2 (sie umfaßt zwei Spalten), und `gridheight` ist 1 (sie umfaßt nur eine Zeile):

```
buildConstraints(constraints, 0, 2, 2, 1, 100, 100);
```

Haben Sie es verstanden? Nun haben Sie die Anordnung der Rahmenbedingungen für alle Komponenten, die Sie in das `GridBagLayout` einfügen möchten, gesetzt. Sie müssen allerdings auch die Rahmenbedingungen jeder Komponente dem Layout-Manager zuordnen und dann jede Komponente in das Panel einfügen. Abbildung 12.8 zeigt das bisherige Ergebnis. Beachten Sie, daß Sie sich hier keine Gedanken um die genauen proportionalen Verhältnisse machen müssen, oder darum, sicherzustellen, daß alles richtig arrangiert ist. Was Sie jetzt im Auge behalten sollten, ist sicherzustellen, daß das Raster funktioniert, daß Sie die richtige Anzahl von Zeilen und Spalten angegeben haben, daß die Spannweiten korrekt sind und daß nichts Auffälliges (Zellen am falschen Platz, sich überlappende Zellen oder ähnliches) zu erkennen ist.

Name	
Password	
OK	

Abbildung 12.8:
GridBagLayout - erster Arbeitsgang

Dritter Schritt: Festlegen der Proportionen

Im nächsten Schritt geht es um die Festlegung der Proportionen von Zeilen und Spalten im Verhältnis zu anderen Zeilen und Spalten. Nehmen wir an, Sie möchten beispielsweise den Platz für die Labels (Name und Paßwort) kleiner als für die Textfelder gestalten. Außerdem möchten Sie die Höhe der Schaltfläche OK im unteren Teil auf lediglich die halbe Höhe der beiden darüber angebrachten Textfelder reduzieren. Verwenden Sie die Rahmenbedingungen `weightx` und `weighty` zur Anordnung der Proportionen der Zelle in Ihrem Layout.

Der einfachste Weg, mit `weightx` und `weighty` umzugehen, ist es, sich ihre Werte als Prozentsätze der Gesamtbreite und -höhe des Panels vorzustellen. Diese können entweder 0 oder eine andere Zahl sein, falls die Gewichtung oder Höhe von einer anderen Zelle gesetzt wurde. Deshalb sollten die Werte für `weightx` und `weighty` aller Komponenten eine Gesamtsumme von 100 ergeben.



Eigentlich sind die Werte für `weightx` und `weighty` keine Prozentsätze; es sind einfach Proportionen - und können einen beliebigen Wert annehmen. Bei der Berechnung der Proportionen werden alle Werte in eine Richtung aufsummiert, so daß jeder einzelne Wert im proportionalen Verhältnis zu dieser Gesamtsumme steht (anders gesagt, durch die Gesamtsumme geteilt, um tatsächlich einen Prozentsatz zu erhalten). Da dieses ganze Verfahren nicht intuitiv ist, ist es wesentlich einfacher, das ganze als Prozentsätze zu betrachten und sicherzustellen, daß das Ergebnis ihrer Summe 100 ist, womit wir auf der sicheren Seite sind.

Welche Zellen erhalten also Werte und welche erhalten 0? Mehrere Zeilen und Spalten umfassende Zellen sollten immer 0 sein, in der Richtung, in der die Komponente sich über mehrere Zellen erstreckt. Darüber hinaus liegt die Entscheidung einfach in der Wahl eines Werts für eine Zelle, wonach dann alle anderen Zellen in dieser Zeile oder Spalte 0 sein sollten.

Schauen wir uns einmal die fünf Aufrufe von `buildConstraints()`, die im vorhergehenden Schritt ausgeführt wurden, an:

```
buildConstraints(constraints, 0, 0, 1, 1, 100, 100); //Label für Namensfeld
buildConstraints(constraints, 1, 0, 1, 1, 100, 100); //Textfeld für den Namen
buildConstraints(constraints, 0, 1, 1, 1, 100, 100); //Label für Paßwort-Feld
buildConstraints(constraints, 1, 1, 1, 1, 100, 100); //Textfeld für das Paßwort
buildConstraints(constraints, 0, 2, 2, 1, 100, 100); //OK-Schaltfläche
```

Die letzten beiden Argumente müssen Sie bei jedem Aufruf von `buildConstraints()` entweder in einen Wert oder 0 umändern. Beginnen wir mit der x-Richtung (die Proportionen der Spalten), welche das zweitletzte Argument in dieser Liste ist.

Wenn Sie sich Abbildung 12.6 noch einmal anschauen (die Illustration des Panels mit dem aufgesetzten Raster), werden Sie feststellen, daß der Umfang der zweiten Spalte bei weitem größer als der der ersten ist. Nehmen wir an, Sie wollen die theoretischen Prozentsätze für diese Spalten auswählen, und nehmen wir weiter an, daß Sie den ersten Prozentsatz mit 10 und den zweiten mit 90 festlegen (alles rein theoretische Annahmen - so sollten auch Sie vorgehen). Diese beiden angenommenen Prozentsätze können Sie nun Zellen zuordnen. Sie können der Zelle nicht

beliebige Werte mit der Schaltfläche OK zuordnen, weil die Zelle beide Spalten einfaßt und deshalb hier nicht mit Prozentsätzen gearbeitet werden kann. Fügen Sie also den ersten beiden Zellen das Label für das Namensfeld und das Textfeld für den Namen hinzu:

```
buildConstraints(constraints, 0, 0, 1, 1, 10, 100); //Label für das Namensfeld
buildConstraints(constraints, 1, 0, 1, 1, 90, 100); //Textfeld für den Namen
```

Und was ist mit den Werten der verbleibenden zwei Zellen, dem Label für das Paßwort-Feld und dem Textfeld dazu? Da mit dem Label für das Namensfeld und dem Textfeld für den Namen die Proportionen der Spalten bereits festgelegt wurden, müssen sie hier nicht neu gesetzt werden. Geben Sie diesen beiden und der Zelle für das OK-Feld die Werte 0:

```
buildConstraints(constraints, 0, 1, 1, 1, 0, 100); //Label für das Paßwort-Feld
buildConstraints(constraints, 1, 1, 1, 1, 0, 100); //Textfeld für das Paßwort
buildConstraints(constraints, 0, 2, 2, 1, 0, 100); //OK-Schaltfläche
```

Beachten Sie in diesem Fall, das der Wert 0 nicht bedeutet, das die Zellenbreite 0 ist. Bei diesen Werten handelt es sich um Proportionen, nicht Pixel-Werte. Eine 0 bedeutet einfach, daß die entsprechende Proportion an anderer Stelle gesetzt wurde - 0 bedeutet eigentlich so viel wie »entsprechend anpassen«.

Die Gesamtsumme aller weightx-Rahmenbedingungen ist jetzt 100, und Sie können sich nun den weighty-Eigenschaften widmen. Hier gibt es drei Zeilen. Wenn Sie einen Blick auf das von Ihnen gezeichnete Raster werfen, sieht es so aus, als ob etwa 20 Prozent auf die Schaltfläche und die restlichen 80 (40 Prozent pro Zeile) auf die Textfelder verteilt sind. Sie müssen den Wert, wie bei den x-Werten, jeweils nur für eine Zelle pro Zeile setzen (die beiden Labels und die Schaltfläche), während alle anderen Zellen als weightx 0 haben.

Hier die endgültigen fünf Aufrufe an buildConstraints() mit den entsprechenden Gewichtungen:

```
buildConstraints(constraints, 0, 0, 1, 1, 10, 40); // Label für das Namensfeld
buildConstraints(constraints, 1, 0, 1, 1, 90, 0); //Textfeld für den Namen
buildConstraints(constraints, 0, 1, 1, 1, 0, 40); //Label für das Paßwort-Feld
buildConstraints(constraints, 1, 1, 1, 1, 0, 0); //Textfeld für das Paßwort
buildConstraints(constraints, 0, 2, 2, 1, 0, 20); //OK-Schaltfläche
```

Abbildung 12.9 zeigt das Ergebnis mit den richtigen Proportionen.

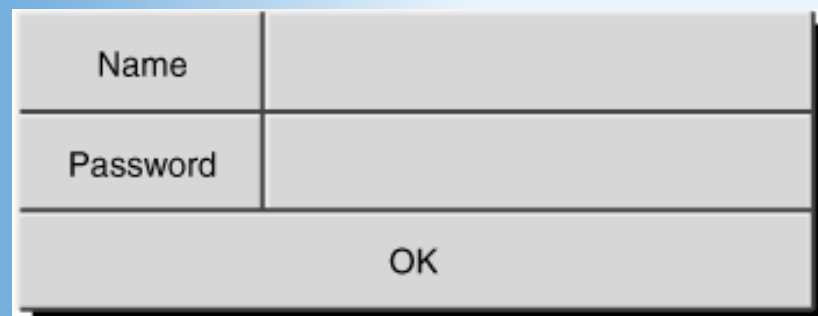


Abbildung 12.9:
GridBagLayout - zweiter Arbeitsgang

Dieser Schritt zielt darauf ab, ein paar einfache Proportionen für die räumliche Anordnung der Zeilen und Zellen am Bildschirm zu erstellen, wodurch Ihnen eine grobe Einschätzung der Größe der verschiedenen Komponenten ermöglicht wird, allerdings sollten Sie an dieser Stelle mit vielen Versuchen und Fehlern rechnen.

Vierter Schritt: Komponenten einfügen und anordnen

Wenn das Layout und die Proportionen entsprechend vorbereitet sind, können Sie die Platzhalter-Schaltflächen durch richtige Label und Textfelder ersetzen. Da Sie hierfür bereits alles vorbereitet haben, sollte das problemlos funktionieren, richtig? Nicht ganz. Abbildung 12.10 zeigt Ihnen das Resultat, wenn Sie dieselben vorherigen Rahmenbedingungen benutzen und die Schaltflächen durch richtige Komponenten ersetzen.

Abbildung 12.10:
GridBagLayout - fast geschafft

Dieses Layout kommt der Sache nahe, aber es sieht sonderbar aus. Die Textfelder sind zu hoch, und die Schaltfläche OK dehnt die Breite der Zelle.

Was jetzt noch fehlt, sind die Rahmenbedingungen, die für die Anordnung der Komponenten in der Zelle sorgen. Es gibt davon zwei: fill und anchor.

Die Rahmenbedingung fill legt für Komponenten, die sich in jede Richtung ausdehnen können (wie Textfelder und Schaltflächen), die Ausdehnungsrichtung fest. fill kann einen von vier als Klassenvariablen in der Klasse GridBagConstraints definierten Werten haben:

- GridBagConstraints.BOTH dehnt die Komponente, so daß sie die Zelle in beiden Richtungen füllt.
- GridBagConstraints.NONE löst die Anzeige der Komponente in ihrer kleinsten Größe aus.
- GridBagConstraints.HORIZONTAL dehnt die Komponente in horizontaler Richtung.
- GridBagConstraints.VERTICAL dehnt die Komponente in vertikaler Richtung.



Denken Sie daran, daß dieses Layout dynamisch ist. Sie richten nicht die tatsächlichen Pixel-Maße von Komponenten ein, sondern bestimmen, in welcher Richtung sich diese Elemente bei einem bestimmten Panel von beliebiger Größe ausdehnen können.

Die Standard-Rahmenbedingung für fill ist für alle Komponenten NONE. Warum also füllen die Textfelder und Label die Zellen? Gehen Sie in Gedanken noch einmal zurück zum Beginn des Codes für dieses Beispiel, wo der init()-Methode folgende Zeile hinzugefügt wurde:

```
constraints.fill = GridBagConstraints.BOTH;
```

Jetzt verstehen Sie ihre Bedeutung. Bei der endgültigen Version dieses Applets entfernen Sie diese Zeile und fügen jeder eigenständigen Komponente die fill-Werte hinzu.

Die zweite für das Aussehen einer Komponente in der Zelle maßgebliche Rahmenbedingung ist anchor. Diese Rahmenbedingung gilt nur für Komponenten, die nicht die gesamte Zelle füllen, und weist das AWT an, wo die Komponente innerhalb der Zelle auszurichten ist. Die möglichen Werte für die anchor-Rahmenbedingung sind GridBagConstraints.CENTER, wodurch die Komponente innerhalb der Zelle sowohl vertikal als auch horizontal zentriert ausgerichtet wird, oder einer von acht Ausrichtungswerten: GridBagConstraints.NORTH, GridBagConstraints.NORTHEAST, GridBagConstraints.EAST, GridBagConstraints.SOUTHEAST, GridBagConstraints.SOUTH, GridBagConstraints.SOUTHWEST, GridBagConstraints.WEST, oder GridBagConstraints.NORTHWEST. Der Standardwert für Anchor ist GridBagConstraints.CENTER.

Diese Rahmenbedingungen setzen Sie genau wie alle anderen: indem Sie die Instanzvariablen im GridBagConstraints-Objekt ändern. Sie können die Definition von buildConstraints() zur Aufnahme von zwei weiteren Argumenten (es handelt sich um int-Werte) ändern, oder Sie können diese einfach innerhalb der init()-Methode setzen. Es empfiehlt sich letzteres.

Behandeln Sie Standard-Werte mit Vorsicht. Denken Sie daran, daß, bedingt durch die erneute Verwendung desselben GridBagConstraints-Objekts für jede Komponente, ein paar Werte übrig bleiben können, nachdem Sie die Bearbeitung einer Komponente abgeschlossen haben. Wenn andererseits der Wert der fill- oder anchor-Eigenschaft

von einem Objekt der gleiche ist wie bei der vorherigen Komponente, haben Sie den Vorteil, das Objekt nicht zurücksetzen zu müssen.

Für ein entsprechendes Beispiel nehmen wir drei Änderungen an den fill- und anchor -Eigenschaften der Komponenten vor:

- Die Labels haben keine fill-Eigenschaften und die anchor-Eigenschaft hat den Wert GridBagConstraints.EAST (so daß sie an der rechten Zellenseite ausgerichtet werden).
- Die Textfelder werden horizontal gefüllt (sie haben die Höhe einer Zeile, und erstrecken sich über die Breite der Zelle).
- Die Schaltfläche hat keine fill-Eigenschaft und wird zentriert ausgerichtet.

Der entsprechende Code hierfür wird Ihnen im folgenden gezeigt; der vollständige Code für das Beispiel ist am Ende dieses Abschnitts aufgeführt. Sie können die vorgenommenen Änderungen erkennen.

Fünfter Schritt: Die letzten Anpassungen

Dieser Schritt wurde der Liste hinzugefügt, da die Erfahrung bei der Erstellung von Layouts mit GridBagLayout gezeigt hat, daß, auch wenn man alle erforderlichen Schritte ausführt, das entstandene Layout sich meist doch nicht so ganz richtig darstellt, und »Herumspielen« mit den verschiedenen Werten bzw. entsprechendes »Zurechtbasteln« der Rahmenbedingungen notwendig ist, um letztendlich das gewünschte Ergebnis zu erhalten. Das ist vollkommen in Ordnung; das Ziel der obigen vier Schritte war es, die endgültige Anordnung so gut wie möglich vorzubereiten und nicht jedes Mal ein perfektes Layout zu entwerfen.

Listing 12.5 zeigt den vollständigen Code für das Layout, das Sie in diesem Abschnitt erstellt haben. Wenn jetzt noch Unklarheiten bestehen, sollten Sie diesen Code zeilenweise durcharbeiten, um ein klares Verständnis der verschiedenen Teile sicherzustellen.

Listing 12.5: Der gesamte Quelltext von NamePass.java

```

1: import java.awt.*;
2:
3: public class NamePass extends java.applet.Applet {
4:
5:     void buildConstraints(GridBagConstraints gbc, int gx, int gy,
6:         int gw, int gh, int wx, int wy) {
7:
8:         gbc.gridx = gx;
9:         gbc.gridy = gy;
10:        gbc.gridwidth = gw;
11:        gbc.gridheight = gh;
12:        gbc.weightx = wx;
13:        gbc.weighty = wy;
14:    }
15:
16:    public void init() {
17:        GridBagLayout gridbag = new GridBagLayout();
18:        GridBagConstraints constraints = new GridBagConstraints();
19:        setLayout(gridbag);
20:
21:        // Der Label für das Textfeld des Namens
22:        buildConstraints(constraints, 0, 0, 1, 1, 10, 40);
23:        constraints.fill = GridBagConstraints.NONE;
24:        constraints.anchor = GridBagConstraints.EAST;
25:        Label label1 = new Label("Name:", Label.LEFT);
26:        gridbag.setConstraints(label1, constraints);
27:        add(label1);
28:
29:        // Das Textfeld für den Namen

```

```

30:      buildConstraints(constraints, 1, 0, 1, 1, 90, 0);
31:      constraints.fill = GridBagConstraints.HORIZONTAL;
32:      TextField tfname = new TextField();
33:      gridbag.setConstraints(tfname, constraints);
34:      add(tfname);
35:
36:      // Der Label für das Textfeld des Paßwortes
37:      buildConstraints(constraints, 0, 1, 1, 1, 0, 40);
38:      constraints.fill = GridBagConstraints.NONE;
39:      constraints.anchor = GridBagConstraints.EAST;
40:      Label label2 = new Label("Password:", Label.LEFT);
41:      gridbag.setConstraints(label2, constraints);
42:      add(label2);
43:
44:      // Das Textfeld des Paßwortes
45:      buildConstraints(constraints, 1, 1, 1, 1, 0, 0);
46:      constraints.fill = GridBagConstraints.HORIZONTAL;
47:      TextField tfpass = new TextField();
48:      tfpass.setEchoCharacter('*');
49:      gridbag.setConstraints(tfpass, constraints);
50:      add(tfpass);
51:
52:      // Die OK-Schaltfläche
53:      buildConstraints(constraints, 0, 2, 2, 1, 0, 20);
54:      constraints.fill = GridBagConstraints.NONE;
55:      constraints.anchor = GridBagConstraints.CENTER;
56:      Button okb = new Button("OK");
57:      gridbag.setConstraints(okb, constraints);
58:      add(okb);
59:  }
60: }

```

Das folgende <APPLET>-Tag wurde verwendet, um dieses Applet zu testen:

```

<applet code="NamePass.class" height=180 width=240>
</applet>

```

Wenn Sie dieses Applet kompilieren, wird der Aufruf der Methode `setEchoCharacter()` in Zeile 48 eine deprecation-Warnung erzeugen, da diese Methode in den Java- Versionen nach 1.02 umbenannt wurde. Sie kann durch die Methode `setEchoChar()` ersetzt werden, wenn Sie ein Applet für die Version 1.2 der Sprache schreiben.

ipadx und ipady

Bevor wir zum Abschluß des `GridBagLayout` kommen (sind wir noch nicht fertig?), müssen noch zwei weitere Rahmenbedingungen erwähnt werden: `ipadx` und `ipady`. Diese beiden Rahmenbedingungen kontrollieren das Auffüllen - hier geht es um den zusätzlichen Raum rund um eine einzelne Komponente. Standardmäßig ist kein zusätzlicher Raum um die Komponenten vorgegeben (was man am besten bei Komponenten, die ihre Zellen füllen, sehen kann).

Mit `ipadx` fügt man jeder Seite der Komponente Platz hinzu, und mit `ipady` fügt man entsprechenden Platz oben und unten ein.

Eckeinsätze (Insets)

Bei der Erstellung eines neuen Layout-Managers (oder bei der Verwendung von `ipadx` und `ipady` in `GridBagLayout`) erzeugte horizontale und vertikale Abstände dienen zur Bestimmung des Platzes zwischen Komponenten in einem Panel. Eckeinsätze hingegen werden zur Festlegung des Rands um das Panel selbst benutzt. Die Klasse `Insets` bietet Eckeinsatzwerte für oben, unten, links und rechts, die dann verwendet werden, wenn das Panel gezeichnet wird.



Eckeinsätze (Insets) dienen zur Bestimmung des Platzes zwischen den Kanten eines Panels und seinen Komponenten.

Um Ihrem Layout einen Eckeinsatz hinzuzufügen, überschreiben Sie entweder die Methode `insets()` oder `getInsets()` in Ihrer Applet-Klasse oder einer anderen Panel- Klasse. Verwenden Sie für Java Version 1.02 `insets()`; und ab der Version 1.1 `getInsets()` . Sie erfüllen den gleichen Zweck, lediglich der Name wurde geändert.

Erstellen Sie mit der Methode `insets()` oder `getInsets()` ein neues Insets-Objekt, wobei der Konstruktor für die Insets-Klasse vier ganzzahlige Werte erwartet, die für die Eckeinsätze oben, unten, links und rechts im Panel stehen. Die Methode `insets()` sollte dann das Insets-Objekt ausgeben. Es folgt ein Beispiel mit Java-1.02-konformem Code zum Einfügen von Eckeinsätzen für ein GridLayout mit den Werten: 10 oben und unten und 30 links und rechts. Abbildung 12.11 zeigt das Ergebnis.

```
public Insets insets() {
return new Insets(10, 30, 10, 30);
}
```



Abbildung 12.11:
Ein Panel mit Eckeinsätzen mit 10 Pixeln oben und unten und 30 Pixeln links und rechts

Zusammenfassung

Abstrakter Expressionismus geht nur so weit, wie Sie das während des heutigen Tages gesehen haben. Layout-Manager benötigen eine gewisse Feinabstimmung für Leute, die eine genauere Kontrolle über die Platzierung von Komponenten, die auf einer Benutzerschnittstelle erscheinen, gewohnt sind.

Sie wissen nun, wie Sie die fünf verschiedenen Layout-Manager verwenden. Wenn Sie mit dem Abstract Windowing Toolkit arbeiten, werden Sie erkennen, daß dieses jede Art von Benutzerschnittstelle durch verschachtelte Container und unterschiedliche Layout-Manager annähern kann.

Sobald Sie die Entwicklung einer Benutzerschnittstelle in Java gemeistert haben, bietet Ihr Programm etwas, was die meisten anderen visuellen Programmiersprachen nicht bieten: eine Benutzerschnittstelle, die ohne Veränderung auf vielen Plattformen läuft.

Um es mit einem oft zitierten Ausspruch zu sagen: »Ich weiß nicht, ob es Kunst ist, aber mir gefällt es.«

Fragen und Antworten

Frage:

Ich mag es überhaupt nicht, mit Layout-Managern zu arbeiten. Entweder sind sie zu einfach von den Möglichkeiten her oder zu kompliziert in der Anwendung (GridBagLayout). Selbst dann, wenn ich viel herumbastle, sehen mein Applets nicht so aus, wie ich mir das vorgestellt habe. Alles was ich will, ist, die Größe meiner Komponenten festzulegen und diese dann an einer bestimmten x,y-Position auf dem Bildschirm auszugeben. Kann ich das?

Antwort:

Es ist möglich, aber sehr problematisch. Das AWT wurde so entworfen, daß die grafische Benutzerschnittstelle eines Programms gleichgut auf unterschiedlichen Plattformen und verschiedenen Bildschirmauflösungen, Schriften, Bildschirmgrößen usw. funktioniert. Wenn Sie sich auf Pixel-Koordinaten verlassen, kann das dazu führen, daß ein Programm auf der einen Plattform gut aussieht und auf anderen nicht zu verwenden ist, da sich einzelne Komponenten überdecken, von den Kanten des Containers abgeschnitten werden usw. Layout-Manager umgehen dieses Problem, indem sie Komponenten dynamisch auf dem Bildschirm plazieren. Obwohl es gewisse Unterschiede im Endergebnis auf den verschiedenen Plattformen gibt, sind diese alles andere als eine Katastrophe.

Immer noch nicht überzeugt? Verwenden Sie null als Layout-Manager und die Methode reshape(), um einer Komponente eine bestimmte Größe zu geben und sie an einer bestimmten Position zu plazieren.

```
setLayout(null);
Button myButton = new Button("OK");
myButton.reshape(10, 10, 30, 15);
```

Mehr Informationen zu reshape() erhalten Sie in der Dokumentation der Klasse Component.

Frage:

Ich habe mir die AWT-Klassen angesehen und ein Paket namens Peer entdeckt. Außerdem wird an vielen Stellen in der API-Dokumentation auf die Peer-Klassen verwiesen. Was bewirken diese Klassen?

Antwort:

Peers sind für die plattformspezifischen Teile des AWT zuständig. Wenn Sie beispielsweise ein AWT-Fenster erstellen, haben Sie eine Instanz der Window- Klasse, die allgemeine Fenstereigenschaften bereitstellt. Daneben gibt es eine Instanz der WindowPeer-Klasse, die ein sehr spezifisches Fenster für diese Plattform - ein Motiv-Fenster unter XWindows, ein Macintosh-Fenster für den Macintosh oder ein Fenster für Windows 95 - erstellt. Diese »Peer«-Klassen handhaben auch die Kommunikation zwischen dem Fenstersystem und dem Java-Fenster selbst. Durch Trennen der allgemeinen Komponenteneigenschaften (AWT-Klassen) von der eigentlichen Systemimplementierung und dem -aussehen (Peer-Klassen) können Sie sich auf das Verhalten Ihrer Java- Anwendung konzentrieren und die plattformspezifischen Einzelheiten der Java-Implementierung überlassen.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).

Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 2

Tag 13

Ereignisverarbeitung in Applets

Mit den Fähigkeiten, die Sie bisher entwickelt haben, können Sie eine grafische Benutzerschnittstelle sehr schön gestalten, allerdings fehlt noch das Gehirn. Sie sieht zwar wie eine funktionierende Benutzerschnittstelle aus - man kann Schaltflächen und andere Elemente der Benutzerschnittstelle bedienen -, allerdings führt dies zu keinerlei Reaktion.

Um eine Benutzerschnittstelle in Java mit Funktionalität zu versehen, müssen Sie lernen, wie Sie ein Programm dazu bringen, auf Ereignisse zu reagieren. Ereignisse sind Methodenaufrufe, die die Fensterverwaltung von Java erzeugt, wenn ein Element der Benutzerschnittstelle verwendet wird. Die große Bandbreite der Ereignisse deckt die Verwendung von Maus und Tastatur ab, inklusive Mausklicks, Mausbewegungen und Tastenanschlägen.

Heute lernen Sie, wie Sie ein Applet dazu bringen, Ereignisse mit den Techniken von Java 1.02 zu verarbeiten, so daß Ihre Programme in jedem Web-Browser ausgeführt werden können, der Java unterstützt. An Tag 21 lernen Sie Ereignisse mit den Techniken von Java 1.2 zu verarbeiten.

Ereignisverarbeitung

Eines der Dinge, die Sie bei der Erstellung Ihres ersten Applets gelernt haben, ist, daß bei der Ausführung des Programms vieles hinter den Kulissen geschieht. Das Fensterverwaltungssystem von Java ruft Methoden, wie z.B. `paint()`, `init()` und `start()`, automatisch auf, wenn diese benötigt werden, ohne daß Sie sich darum kümmern müssen.

Wie die Applet-Programmierung schließt die Ereignisbehandlung Methoden ein, die automatisch aufgerufen werden, wenn eine Aktion eines Benutzers ein Ereignis auslöst.

Ereignisarten

Ein Ereignis wird als Reaktion auf nahezu alles, was ein Benutzer während des Lebenszyklus eines Java-Programms tun kann, erzeugt. Jede Bewegung der Maus, ein Klick auf eine Schaltfläche oder ein Tastenanschlag erzeugen ein Ereignis.

In Ihren Programmen müssen Sie sich nicht um alle Ereignisse kümmern, die auftreten könnten. Statt dessen verarbeiten Sie nur die Ereignisse, auf die Ihr Programm reagieren soll. Wenn z.B. der Benutzer mit der Maus innerhalb des Applet-Fensters klickt oder eine Taste drückt, dann wollen Sie eventuell, daß Ihr Programm eine bestimmte Aktion als Reaktion auf das Ereignis ausführt.

Die folgenden Ereignisse sind einige der Ereignisse, die Sie in Ihren eigenen Programmen behandeln können. (In der Auflistung finden Sie die englische Bezeichnung für die einzelnen Ereignisse, wie sie unter Java üblich sind. In Klammern dahinter befindet sich jeweils eine Beschreibung der Aktion, die das Ereignis auslöst):

- *Mausklicks.* `mouseDown` (Maustaste gedrückt), `mouseUp` (Maustaste wird wieder losgelassen, nachdem sie gedrückt wurde).
- *Mausbewegungen.* `mouseEnter` und `mouseExit` (der Mauszeiger betritt bzw. verläßt den Bereich einer Komponente der Benutzerschnittstelle), `mouseMove` (die Maus wird innerhalb des

Applet-Fensters bewegt), mouseDrag (die Maus wird innerhalb des Applet-Fensters bei gedrückter Maustaste bewegt).

- *Tastenanschläge.* keyDown (eine Taste wird gedrückt), keyUp (eine zuvor gedrückte Taste wird losgelassen).
- *Ereignisse der Benutzerschnittstelle.* Anklicken einer Schaltfläche, Scrollen einer Bildlaufleiste, Öffnen von Menüs usw.

Die Methode `handleEvent()`

Die Ereignisbehandlung ist der Bereich, in dem sich Java zwischen Java 1.02 und der aktuellen Version Java 1.2 verändert hat. Die Ereignisse selbst werden nahezu gleich erzeugt und durchlaufen das System auch in nahezu der gleichen Weise, unabhängig davon, mit welcher Version der Sprache Sie ein Programm schreiben. Der Unterschied liegt darin, wie Ereignisse empfangen und verarbeitet werden.

In Java 1.02 durchlaufen alle Ereignisse, die während des Lebenszyklus Ihres Java- Programms erzeugt werden, dieses Programm und werden von einer Methode namens `handleEvent()` behandelt. Diese Methode ist in der Klasse `Component` definiert, von der die Klasse `java.applet.Applet` abgeleitet ist. Dadurch steht diese Methode in allen Ihren Applets zur Verfügung.

Wenn ein Ereignis an die Methode `handleEvent()` in Ihrem Applet geschickt wird, ruft diese eine speziellere Methode zur Verarbeitung des jeweiligen Ereignisses auf. Einige dieser spezielleren Methoden sind `mouseDown()`, `mouseUp()` und `keyDown()`.

Um ein Ereignis in Ihren Applets verarbeiten zu können, überschreiben Sie eine dieser spezielleren Methoden zur Ereignisbehandlung. Anschließend wird, sobald dieses Ereignis auftritt, Ihre Methode aufgerufen. Sie könnten z.B. die Methode `mouseDown()` mit dem Verhalten, das für die Anzeige einer Meldung im Applet-Fenster sorgt, überschreiben. Wenn nun ein `mouseDown`-Ereignis auftritt, wird diese Nachricht angezeigt.

Mausklicks behandeln

Zu den gängigsten Ereignissen in einem Applet zählen die Mausklicks. Mausklickereignisse finden statt, wenn ein Benutzer eine beliebige Position innerhalb des Applets anklickt. Sie können Mausklicks für einfache Dinge einsetzen, z.B. um Klänge im Applet an- oder auszuschalten, um zum nächsten Dia in einer Präsentation zu gelangen oder den Bildschirm zu leeren und neu zu beginnen. Mausklicks lassen sich auch in Verbindung mit Mausbewegungen benutzen, wenn in einem Applet komplexere Interaktionen durchgeführt werden sollen.

`mouseDown` und `mouseUp`

Wenn Sie einmal mit der Maus klicken, erzeugt AWT 1.02 zwei separate Ereignisse: ein `mouseDown`-Ereignis, wenn die Maustaste gedrückt wird, und ein `mouseUp`-Ereignis, wenn sie wieder losgelassen wird. Warum zwei verschiedene Ereignisse für eine einzige Mausektion? Weil Sie vielleicht für »Down« und »Up« verschiedene Dinge festlegen möchten. Als Beispiel soll ein Pull-down-Menü dienen: Wenn Sie die Maustaste drücken, wird das Menü angezeigt, sobald Sie die Maustaste über einer Option loslassen, wird diese ausgewählt - dazwischen wird die Maus gezogen (doch dazu später mehr). Wenn Sie nur ein Ereignis für beide Aktionen (`mouseDown` und `mouseUp`) zur Verfügung hätten, ließe sich diese Art von Benutzerinteraktion nicht implementieren.

Die Behandlung von Mausereignissen in einem Applet ist einfach. Sie überschreiben lediglich die entsprechende Methodendefinition. Diese Methode wird aufgerufen, wenn das betreffende Ereignis

stattfindet. Im folgenden finden Sie ein Beispiel der Methodensignatur für ein mouseDown-Ereignis:

```
public boolean mouseDown(Event evt, int x, int y) {
    ...
}
```

Die mouseDown()-Methode (und auch die mouseUp()-Methode) erhält drei Parameter: das Ereignis und die x- und y-Koordinaten, an denen das mouseDown- oder mouseUp-Ereignis stattfindet.

Das Ereignisargument evt ist eine Instanz der Klasse Event. Alle Systemereignisse erzeugen eine Instanz der Event-Klasse, die Informationen darüber enthält, wo und wann ein Ereignis stattfindet, um welches Ereignis es sich handelt und andere Informationen, die über ein Ereignis von Interesse sind. Gelegentlich ist es sinnvoll, für ein Event-Objekt eine Referenz zu erstellen, wie Sie später in dieser Lektion noch erfahren werden.

Die x- und y-Koordinaten des Ereignisses werden von den x- und y-Argumenten weitergegeben und sind besonders nützlich, weil Sie auf ihrer Grundlage genau festlegen können, wo der Mausklick erfolgt ist. Wenn z.B. das mouseDown-Ereignis über einer grafischen Schaltfläche stattgefunden hat, könnten Sie diese Schaltfläche aktivieren. Beachten Sie, daß Sie auf die x- und y-Koordinaten innerhalb des Event-Objekts direkt zugreifen können; sie werden als separate Variablen weitergeleitet, wodurch sie einfacher zu behandeln sind.

Im folgenden finden Sie eine einfache Methode, welche Informationen über ein mouseDown-Ereignis zum Zeitpunkt der Ausführung ausgibt:

```
public boolean mouseDown(Event evt, int x, int y) {
    System.out.println("Mouse down at " + x + ", " + y);
    return true;
}
```

Indem Sie die Methode in das Applet einfügen, wird diese Meldung jedesmal ausgegeben, wenn der Benutzer mit der Maus auf das Applet klickt. Das AWT-System ruft jede einzelne Methode auf, wenn das betreffende Ereignis stattfindet.



Anders als bei Java-Anwendungen, wo Ausgaben von System.out.print() am Bildschirm erfolgen, variiert die Ausgabe in Applets von System zu System und von Browser zu Browser. Der Appletviewer zeigt die Zeile im selben Fenster an, in dem auch die Anweisung gegeben wurde. Netscape verfügt über ein spezielles Fenster namens Java-Konsole, das sichtbar sein muß, damit diese Ausgabe erscheint. Der Internet Explorer leitet Java-Ausgaben in eine separate Datei. Testen Sie in den jeweiligen Umgebungen, wo die Ausgabe von Java-Applets erfolgt.

Beachten Sie, daß diese Methode, anders als die bisher erläuterten Systemmethoden, anstelle von void einen booleschen Wert ausgibt. Die Bedeutung dieses Unterschieds wird morgen klar, wenn Sie Benutzeroberflächen erstellen und Eingaben für diese Oberflächen definieren. Mit einer Ereignisbehandlung, die true oder false ausgibt, bestimmen Sie, ob eine bestimmte Komponente der Benutzeroberfläche in ein Ereignis eingreifen kann oder ob es an die übergeordnete Komponente im gesamten System abzugeben ist. Die allgemeine Regel lautet, daß Methoden, die Ereignisse behandeln, true zurückgeben sollten. Wenn die Methode aus irgendeinem Grund nicht auf das Ereignis reagiert, sollte false zurückgegeben werden, damit die anderen Komponenten im System die Chance erhalten, dieses Ereignis zu verarbeiten. In den meisten Beispielen der heutigen Lektion werden Sie auf einfache Ereignisse reagieren, weshalb hier immer true zurückgegeben wird.

Die zweite Hälfte des Mausklicks ist die `mouseUp`-Methode, die aufgerufen wird, sobald die Maustaste losgelassen ist. Um ein `mouseUp`-Ereignis zu behandeln, fügen Sie die `mouseUp()`-Methode in das Applet ein: `mouseUp()` sieht genauso aus wie `mouseDown()`:

```
public boolean mouseUp(Event evt, int x, int y) {
    ....
}
```

Beispiel: Punkte

In diesem Abschnitt erstellen Sie ein Beispiel-Applet, das das Ereignismodell 1.02 zur Behandlung von Mausereignissen - insbesondere `mouseDown`-Ereignissen - verwendet. Das Spots-Applet beginnt mit einem leeren Bildschirm und wartet dann ab. Wird die Maus in diesem Bildschirm geklickt, erscheint ein blauer Punkt. Sie können in diesem Bildschirm bis zu zehn Punkte anordnen. Abbildung 13.1 zeigt das Spots-Applet.

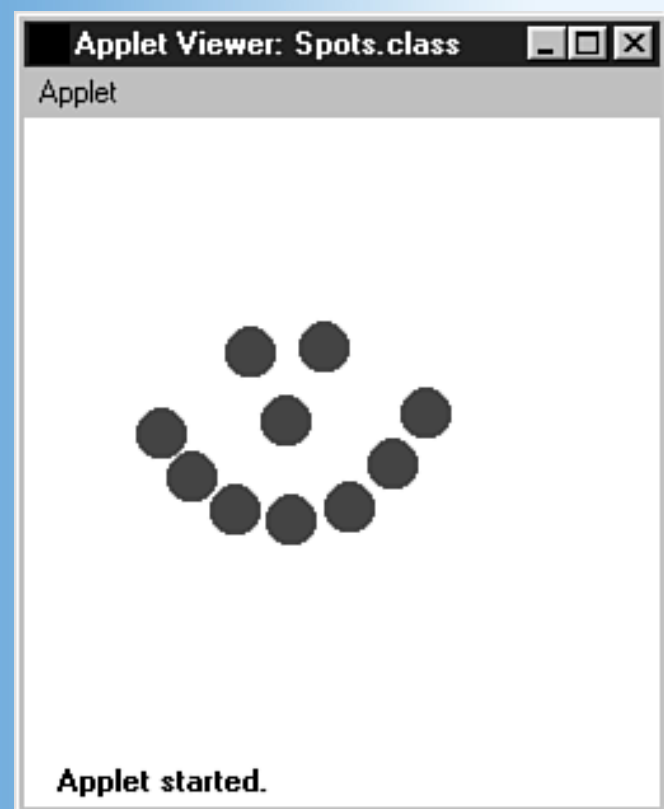


Abbildung 13.1:
Das Spots-Applet

Wir erstellen dieses Applet nun von Anfang an. Beginnen Sie mit der Klassendefinition:

```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Event;

public class Spots extends java.applet.Applet {

    final int MAXSPOTS = 10;
    int xspots[] = new int[MAXSPOTS];
    int yspots[] = new int[MAXSPOTS];
    int currspots = 0;
```

}

In dieser Definition werden drei andere AWT-Klassen verwendet: Graphics, Color und Event. Event muß in jedes Applet importiert werden. Die Klasse hat vier Instanzvariablen: eine Konstante, um die Höchstzahl der zu zeichnenden Punkte festzulegen, zwei Arrays, um die x- und y-Koordinaten der bereits gezeichneten Punkte zu speichern, und eine Ganzzahl, um die Nummer des aktuellen Punkts zu verfolgen.



In der Definition der Event-Klasse ist implements Runnable nicht enthalten. Wie Sie im weiteren Verlauf der Arbeit mit diesem Applet noch sehen werden, verfügt es auch nicht über eine run()-Methode. Warum nicht? Weil das Applet im eigentlichen Sinn selbst keine Aufgaben durchführt, es wartet im Grunde nur auf Benutzereingaben und reagiert darauf. Es besteht keine Notwendigkeit für Threads, wenn ein Applet nicht die ganze Zeit über selbst aktiv ist.

Jetzt schreiben Sie die init()-Methode, die aus einer Zeile besteht, um als Hintergrundfarbe Weiß zu definieren:

```
public void init() {
    setBackground(Color.white);
}
```

Der Hintergrund wird hier und nicht in paint() definiert, weil paint() wiederholt aufgerufen wird, sobald ein neuer Punkt hinzukommt. Da Sie den Hintergrund aber nur einmal einstellen möchten, würde eine Einbindung in die paint()-Methode den Ablauf verlangsamen.

Die Hauptaktion dieses Applets findet in der mouseDown()-Methode statt. Sie fügen jetzt eine solche Methode ein:

```
public boolean mouseDown(Event evt, int x, int y) {
    if (currspots < MAXSPOTS) {
        addspot(x,y);
        return true;
    }
    else {
        System.out.println("Too many spots.");
        return false;
    }
}
```

Findet der Mausklick statt, prüft die mouseDown()-Methode, ob weniger als zehn Punkte vorhanden sind. Trifft dies zu, ruft sie die addspot()-Methode (die Sie im Anschluß schreiben) auf und gibt true zurück (mouseDown-Ereignis wurde aufgegriffen und behandelt). Andernfalls wird eine Fehlermeldung aus- und false zurückgegeben. Was bewirkt addspot()? Diese Methode fügt die Koordinaten des Punkts in die Arrays ein, in denen die Koordinaten gespeichert werden, erhöht die currspots-Variable und ruft dann repaint() auf:

```
void addspot(int x, int y) {
    xspots[currspots] = x;
    yspots[currspots] = y;
    currspots++;
    repaint();
}
```

Vielleicht fragen Sie sich, warum man neben dem aktuellen Punkt auch alle bereits aktivierten Punkte verfolgen muß. Der Grund liegt an `repaint()`: Jedesmal, wenn der Bildschirm neu gezeichnet wird, müssen zusätzlich zum letzten auch alle alten Punkte ausgegeben werden. Andernfalls würde immer nur jeweils der aktuelle Punkt ohne die alten Punkte erscheinen.

Nun wenden wir uns der `paint()`-Methode zu:

```
public void paint(Graphics g) {
    g.setColor(Color.blue);
    for (int i = 0; i < currspots; i++) {
        g.filloval(xspots[i] - 10, yspots[i] - 10, 20, 20);
    }
}
```

Innerhalb von `paint()` gehen Sie die in den `xspots`- und `yspots`-Arrays gespeicherten Punkte in einer Schleife durch, damit einer nach dem anderen gezeichnet wird (leicht nach rechts oben gerückt, damit der Punkt rund um den Mauszeiger und nicht unterhalb rechts ausgegeben wird).

Das war's schon! Damit haben Sie ein Applet geschrieben, das Mausklicks behandelt. Den Rest überlasse ich Ihnen. Sie müssen das entsprechende Verhalten für `mouseDown()` oder `mouseUp()` einfügen, damit die Ereignisse abgewickelt werden. Den kompletten Code für das Spots-Applet finden Sie in Listing 13.1.

Listing 13.1: Der gesamte Quelltext von Spots.java

```
1: import java.awt.Graphics;
2: import java.awt.Color;
3: import java.awt.Event;
4:
5: public class Spots extends java.applet.Applet {
6:
7:     final int MAXSPOTS = 10;
8:     int xspots[] = new int[MAXSPOTS];
9:     int yspots[] = new int[MAXSPOTS];
10:    int currspots = 0;
11:
12:    public void init() {
13:        setBackground(Color.white);
14:    }
15:
16:    public boolean mouseDown(Event evt, int x, int y) {
17:        if (currspots < MAXSPOTS) {
18:            addspot(x,y);
19:            return true;
20:        }
21:        else {
22:            System.out.println("Too many spots.");
23:            return false;
24:        }
25:    }
26:
27:    void addspot(int x,int y) {
28:        xspots[currspots] = x;
29:        yspots[currspots] = y;
30:        currspots++;
    }
```



```

31:         repaint();
32:     }
33:
34:     public void paint(Graphics g) {
35:         g.setColor(Color.blue);
36:         for (int i = 0; i < currspots; i++) {
37:             g.filloval(xspots[i] - 10, yspots[i] - 10, 20, 20);
38:         }
39:     }
40: }

```

Sie können dieses Applet in einer Webseite mit dem folgenden HTML-Code laden:

```

<applet code="Spots.class" height=250 width=250>
</applet>

```

Doppelklicks

Was ist zu tun, wenn Sie nicht nur an einfachen Mausklicks interessiert sind? Wie können Sie doppelte oder dreifache Mausklicks behandeln? Die Event-Klasse von Java enthält eine Variable namens `clickCount` zum Verfolgen dieser Informationen. `clickCount` ist eine Ganzzahl, welche die Anzahl aufeinanderfolgender Mausklicks wiedergibt, die stattgefunden haben (was als aufeinanderfolgend interpretiert wird, ist meist im Betriebssystem oder der Maus-Hardware festgelegt). Wenn Sie in den Applets mehrere Mausklicks behandeln möchten, können Sie diesen Wert innerhalb der `mouseDown()`-Methode wie folgt testen:

```

public boolean mouseDown(Event evt, int x, int y) {
    switch (evt.clickCount) {
        case 1: // einzel-Klick
        case 2: // doppel-Klick
        case 3: // triple-Klick
            ....
    }
}

```

Einen wichtigen Punkt gibt es zu beachten, wenn Sie auf Doppel- oder Tripel-Klicks prüfen: `mouseDown()` wird für jeden Klick mit der Maus aufgerufen. Nehmen Sie das folgende Code-Beispiel:

```

public boolean mouseDown(Event evt, int x, int y) {
    system.out.println("Click count: " + evt.clickCount);
    return false;
}

```

Wenn Sie diese Methode in ein Applet einfügen, wird folgendes angezeigt, wenn mit der Maus dreimal geklickt wird:

```

Click count: 1
Click count: 2
Click count: 3

```

Wie Sie etwas später am heutigen Tag lernen werden, erzeugen einige Komponenten ein action-Ereignis, wenn doppelt auf diese Komponenten geklickt wird. Aus diesem Grund ist es nicht immer nötig, die `mouseDown()`-Methode zu verwenden, um zwischen einfachen und doppelten Klicks, die von einer Komponente erzeugt wurden, zu unterscheiden.

Mausbewegungen behandeln

Jedesmal, wenn die Maus um ein Pixel in eine Richtung bewegt wird, wird ein Mausbewegungsereignis erzeugt. Wir unterscheiden zwei Mausbewegungsereignisse: das Ziehen der Maus mit gedrückter Maustaste und die einfache Bewegung des Mauszeigers ohne Drücken einer Maustaste.

Darüber hinaus werden weitere Ereignisse beim Eintritt des Mauszeigers in den Bereich des Applets, einer Komponente oder eines Containers auf dem Applet bzw. bei dessen Verlassen erzeugt.

Für jedes dieser Ereignisse werden spezielle Methoden definiert, die diese Ereignisse aufgreifen, ebenso wie die Methoden `mouseDown()` und `mouseUp()` für die Verarbeitung von Mausklicks.

mouseDrag und mouseMove

Um Mausbewegungsereignisse zu behandeln, verwenden Sie die Methoden `mouseDrag()` und `mouseMove()`.

Die `mouseMove()`-Methode zur Behandlung einfacher Mausbewegungen ohne gedrückte Maustaste, ist den Mausklick-Methoden sehr ähnlich:

```
public boolean mouseMove(Event evt, int x, int y) {
    ...
}
```

Die `mouseDrag()`-Methode behandelt Mausbewegungen, die mit gedrückter Maustaste durchgeführt werden (eine komplette Ziehbewegung besteht aus einem `mouseDown`- Ereignis, einer Reihe von `mouseDrag`-Ereignissen für jedes Pixel, um das sich die Maus bewegt, und einem `mouseUp`-Ereignis, wenn die Maustaste losgelassen wird). Die `mouseDrag()`-Methode sieht wie folgt aus:

```
public boolean mouseDrag(Event evt, int x, int y) {
    ...
}
```

Beachten Sie, daß die Argumente in `mouseMove()` und `mouseDrag()` die neue Mausposition kennzeichnen und nicht den Ausgangspunkt der Bewegung.

mouseenter und mouseExit

Die `mouseenter()`- und `mouseExit()`-Methoden werden aufgerufen, wenn der Mauszeiger den Applet-Bereich oder einen Bestandteil davon »betritt« bzw. verläßt. Sowohl `mouseenter()` als auch `mouseExit()` haben ähnliche Signaturen wie die Mausklick-Methoden. Sie verfügen über drei Argumente: das Event-Objekt und die x- und y-Koordinaten des Punkts, an dem der Mauszeiger über dem Applet-Bereich steht oder an dem er den Applet-Bereich verläßt. Die folgenden Beispiele zeigen die Signaturen für `mouseenter()` und `mouseExit()`:

```
public boolean mouseEnter(Event evt, int x, int y) {
    ...
}
```

```
public boolean mouseExit(Event evt, int x, int y) {
    ...
}
```

Beispiel: Linien zeichnen

In diesem Abschnitt schreiben Sie ein Applet, mit dem gerade Linien am Bildschirm durch Ziehen der Maus vom Anfangs- zum Endpunkt erstellt werden. Das ausgeführte Applet ist in Abbildung 13.2 dargestellt.

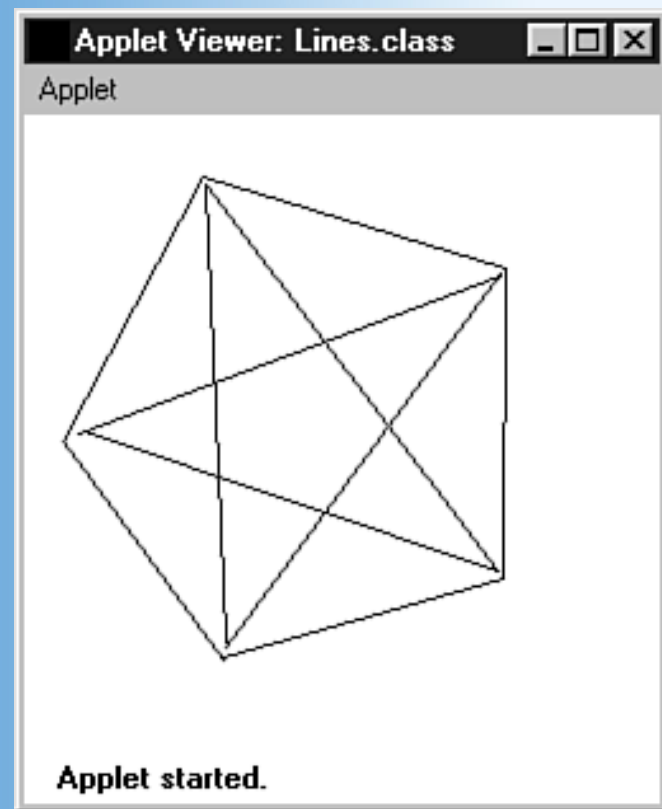


Abbildung 13.2:
Linien zeichnen

Wie beim Spots-Applet (auf dem dieses Applet basiert) beginnen Sie mit der Definition und arbeiten dann die einzelnen Schritte durch, indem Sie die betreffenden Methoden für das Applet hinzufügen. Im folgenden finden Sie die Klassendefinition für das Lines-Applet, mit einigen Instanzvariablen und einer einfachen init()-Methode:

```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Event;
import java.awt.Point;

public class Lines extends java.applet.Applet {

    final int MAXLINES = 10;
    Point starts[] = new Point[MAXLINES]; // Startpunkte
    Point ends[] = new Point[MAXLINES];   // Endpunkte
    Point anchor;    // Start der aktuellen Linie
    Point currentpoint; // Ende der aktuellen Linie
    int currline = 0; // Anzahl der Linien

    public void init() {
        setBackground(Color.white);
    }
}
```

}

Dieses Applet verfügt im Vergleich zum Spots-Applet über einige Zusätze. Im Gegensatz zu `Spots`, das die einzelnen ganzzahligen Koordinaten verfolgt, werden hier `Point`- Objekte verfolgt. Diese Punkte stellen eine x- und y-Koordinate dar, die in einem Objekt eingekapselt ist. Um Punkte zu handhaben, importieren Sie die `Point`-Klasse (`java.awt.Point`) und richten eine Reihe von Instanzvariablen ein, die diese Punkte enthalten:

- Das `starts`-Array nimmt die Punkte auf, die den Anfang der bereits bezeichneten Linien darstellen.
- Das `ends`-Array nimmt die Endpunkte dieser Linien auf.
- `Anchor` nimmt den Anfangspunkt der momentan gezeichneten Linie auf.
- `Currentpoint` nimmt den Endpunkt der momentan gezeichneten Linie auf.
- `Currline` nimmt die aktuelle Zahl der Linien auf (um sicherzustellen, daß `maxlines` nicht überschritten wird).

In der `init()`-Methode wird schließlich festgelegt, daß der Hintergrund des Applets weiß sein soll (wie auch in `Spots`).

Die drei wichtigen Ereignisse dieses Applets sind `mouseDown()`, um den Ankerpunkt für die aktuelle Linie zu setzen, `mouseDrag()`, um die aktuelle Linie während des Zeichnens zu animieren, und `mouseUp()`, um den Endpunkt für die neue Linie zu definieren. Da Sie über Instanzvariablen für diese Werte verfügen, müssen Sie nur noch die richtigen Variablen in die richtigen Methoden einfügen. Im folgenden setzen Sie mit `mouseDown()` den Ankerpunkt (aber nur wenn die maximale Anzahl der Linien nicht überschritten wurde):

```
public boolean mouseDown(Event evt, int x, int y) {
    if (currline < MAXLINES) {
        anchor = new Point(x,y);
        return true;
    }
    else {
        System.out.println("Too many lines.");
        return false;
    }
}
```

Während die Maus gezogen wird, um die Linie zu zeichnen, animiert das Applet die momentan gezeichnete Linie. Durch Ziehen der Maus bewegt sich die neue Linie vom Ankerpunkt zur Spitze des Mauszeigers. Das `mouseDrag()`-Ereignis enthält den jeweils aktuellen Punkt, auf dem sich die Maus bewegt, deshalb wird diese Methode benutzt, um den aktuellen Punkt zu verfolgen (und bei jeder Bewegung nachzuzeichnen, um die Linie zu »animieren«). Falls Sie die maximale Linienanzahl überschritten haben, läßt sich diese Arbeit nicht ausführen. Im folgenden finden Sie die `mouseDrag()`-Methode, welche alle diese Aufgaben ausführt:

```
public boolean mouseDrag(Event evt, int x, int y) {
    if (currline < MAXLINES) {
        currentpoint = new Point(x,y);
        repaint();
        return true;
    }
    else return false;
}
```

Die neue Linie wird erst beim Loslassen der Maustaste in die Arrays der alten Linien aufgenommen. Hier

wird mit `mouseUp()` sichergestellt, daß die Höchstzahl der Linien nicht überschritten wurde, bevor die `addline()`-Methode (wird weiter unten beschrieben) aufgerufen wird:

```
public boolean mouseUp(Event evt, int x, int y) {
    if (currline < MAXLINES) {
        addline(x,y);
        return true;
    }
    else return false;
}
```

In der `addline()`-Methode werden die Arrays für Anfangs- und Endpunkte aktualisiert und das Applet wird nachgezeichnet, um die jeweils neue Linie zu berücksichtigen:

```
void addline(int x,int y) {
    starts[currline] = anchor;
    ends[currline] = new Point(x,y);
    currline++;
    currentpoint = null;
    anchor = null;
    repaint();
}
```

Beachten Sie, daß in dieser Methode auch `currentpoint` und `anchor` auf `null` gesetzt werden. Warum? Weil der Zeichenvorgang für die aktuelle Linie beendet ist. Indem Sie diese Variablen auf `null` setzen, können Sie diesen Wert in der `paint()`-Methode prüfen, um herauszufinden, ob Sie eine aktuelle Linie zeichnen müssen.

Die Ausgabe des Applets bedeutet, daß alle alten Linien, die in den `starts`- und `ends`- Arrays gespeichert sind, zusätzlich zur jeweils aktuellen Linie (deren Endpunkte in `anchor` bzw. `currentpoint` stehen) gezeichnet werden. Um die Animation der aktuellen Linie für den Benutzer gut sichtbar darzustellen, wird sie in Blau ausgegeben. Im folgenden finden Sie die `paint()`-Methode für das Lines-Applet.

```
public void paint(Graphics g) {
    // Bestehende Linien zeichnen
    for (int i = 0; i < currline; i++) {
        g.drawLine(starts[i].x, starts[i].y,
            ends[i].x, ends[i].y);
    }

    // Aktuelle Linie zeichnen
    g.setColor(Color.blue);
    if (currentpoint != null)
        g.drawLine(anchor.x, anchor.y,
            currentpoint.x, currentpoint.y);
}
```

Wenn Sie jeweils die aktuelle Linie in `paint` einbinden, können Sie vorab testen, ob `currentpoint` einen Wert verschieden von `null` hat. In diesem Fall wird gerade keine Linie im Applet gezeichnet, deshalb besteht kein Grund, eine Linie auszugeben, die nicht existiert. Durch Testen von `currentpoint` (und indem `currentpoint` in der `addline()` -Methode auf `null` gesetzt wird), können Sie die Ausgabe auf das beschränken, was nötig ist. Mit nur 60 Codezeilen und einigen einfachen Methoden haben Sie eine einfache Zeichenanwendung für den Web-Browser entwickelt. Listing 13.2 zeigt den kompletten Quelltext des Lines-Applets, in dem die einzelnen Teile zusammengefaßt werden.

Listing 13.2: Der komplette Quelltext von Lines.java

```
1: import java.awt.Graphics;
2: import java.awt.Color;
3: import java.awt.Event;
4: import java.awt.Point;
5:
6: public class Lines extends java.applet.Applet {
7:
8:     final int MAXLINES = 10;
9:     Point starts[] = new Point[MAXLINES]; // Startpunkte
10:    Point ends[] = new Point[MAXLINES];    // Endpunkte
11:    Point anchor;    // Start der aktuellen Linie
12:    Point currentpoint; // Ende der aktuellen Linie
13:    int currline = 0; // Anzahl der Linien
14:
15:    public void init() {
16:        setBackground(Color.white);
17:    }
18:
19:    public boolean mouseDown(Event evt, int x, int y) {
20:        if (currline < MAXLINES) {
21:            anchor = new Point(x,y);
22:            return true;
23:        }
24:        else {
25:            System.out.println("Too many lines.");
26:            return false;
27:        }
28:    }
29:
30:    public boolean mouseUp(Event evt, int x, int y) {
31:        if (currline < MAXLINES) {
32:            addline(x,y);
33:            return true;
34:        }
35:        else return false;
36:    }
37:
38:    public boolean mouseDrag(Event evt, int x, int y) {
39:        if (currline < MAXLINES) {
40:            currentpoint = new Point(x,y);
41:            repaint();
42:            return true;
43:        }
44:        else return false;
45:    }
46:
47:    void addline(int x,int y) {
48:        starts[currline] = anchor;
49:        ends[currline] = new Point(x,y);
50:        currline++;
```

```

51:         currentpoint = null;
52:         anchor = null;
53:         repaint();
54:     }
55:
56:     public void paint(Graphics g) {
57:
58:         // Bestehende Linien zeichnen
59:         for (int i = 0; i < currline; i++) {
60:             g.drawLine(starts[i].x, starts[i].y,
61:                 ends[i].x, ends[i].y);
62:         }
63:
64:         // Aktuelle Linie zeichnen
65:         g.setColor(Color.blue);
66:         if (currentpoint != null)
67:             g.drawLine(anchor.x, anchor.y,
68:                 currentpoint.x, currentpoint.y);
69:     }
70: }

```

Sie können dieses Applet testen, indem Sie den folgenden HTML-Code verwenden:

```

<applet code="Lines.class" height=250 width=250>
</applet>

```

Behandlung von Tastaturereignissen

Ein Tastaturereignis wird erzeugt, wenn ein Benutzer eine Taste auf der Tastatur drückt. Wenn Sie Tastaturereignisse verwenden, können Sie die Werte jener Tasten ermitteln, die der Benutzer zur Durchführung einer Aktion gedrückt hat, oder von den Benutzern Ihres Applets eine Zeicheneingabe erhalten.

Damit ein Tastaturereignis von einer Komponente empfangen werden kann, muß diese Komponente den Eingabefokus besitzen. Anders ausgedrückt muß diese Komponente diejenige auf der Benutzerschnittstelle sein, die aktuell für die Benutzereingaben ausgewählt ist. Sie lernen über den Eingabefokus später am heutigen Tag mehr, wenn Sie mit Fokusereignissen arbeiten. Der Eingabefokus läßt sich am einfachsten verstehen, wenn Sie sich eine Benutzerschnittstelle vorstellen, die über diverse Texteingabefelder verfügt. Die Eingabemarke blinkt in dem Textfeld, das den Fokus hat, und ein Benutzer kann über die Tastatur Text in dieses Textfeld eingeben. Kein anderes Textfeld kann Text empfangen, bis es den Eingabefokus zugeteilt bekommt. Alle Komponenten, inklusive der Container, können den Fokus erhalten.

Um ausdrücklich festzulegen, daß eine Komponente den Fokus hat, kann die Methode `requestFocus()` der Komponente ohne Argumente aufgerufen werden. Die folgende Anweisung erteilt einer Schaltfläche (einem Button-Objekt mit dem Namen `quit`):

```
quit.requestFocus();
```

Sie können einem Applet-Fenster den Fokus erteilen, indem Sie dessen `requestFocus()` -Methode aufrufen.

keyDown und keyUp

Um ein Tastaturereignis zu behandeln, verwenden Sie die Methode `keyDown()`:

```
public boolean keyDown(Event evt, int key) {
    ...
}
```

Die Tasten, die von den `keyDown`-Ereignissen erzeugt und als Tastenargument in `keyDown()` weitergeleitet werden, sind Ganzzahlen, die eindeutige Zeichenwerte darstellen, zu denen sowohl die alphanumerischen Zeichen, als auch die Funktionstasten, Tabstopps, Absatzzeichen usw. gehören. Wenn Sie diese als Zeichen verwenden möchten (etwa für die Ausgabe), müssen Sie diese wie folgt als Zeichen definieren:

```
currentchar = (char)key;
```

Im folgenden finden Sie ein einfaches Beispiel für eine `keyDown()`-Methode, die im Grunde lediglich jene Taste zurückgibt, deren Unicode und Zeichendarstellungen Sie soeben eingegeben haben (es macht Spaß zu beobachten, welche Tasten welche Werte erzeugen):

```
public boolean keyDown(Event evt, int key) {
    System.out.println("ASCII value: " + key);
    System.out.println("Character: " + (char)key);
    return true;
}
```

Ebenso wie bei Mausklicks verfügt auch jedes `keyDown`-Ereignis über ein entsprechende `keyUp`-Ereignis. Um `keyUp`-Ereignisse zu behandeln, verwenden Sie die folgende `keyUp()`-Methode:

```
public boolean keyUp(Event evt, int key) {
    ...
}
```

Standardtasten

Die `Event`-Klasse bietet verschiedene Klassenvariablen, die sich auf mehrere nicht alphanumerische Standardtasten, z.B. die Pfeiltasten, beziehen. Werden diese Tasten in Ihrem Applet benutzt, können Sie den Code übersichtlicher gestalten, indem Sie in Ihrer `keyDown()`-Methode für diese Tasten Namen anstelle von numerischen Werten verwenden. Um z.B. zu prüfen, ob die (½)-Taste gedrückt wurde, können Sie folgenden Code schreiben:

```
if (key == Event.UP) {
    ...
}
```

Da diese Klassenvariablen ganzzahlige Werte enthalten, können Sie auch die `switch`-Anweisung verwenden, um diese zu testen.



Falls Sie Solaris 7 einsetzen und eine Komponente nicht auf die Pfeiltasten reagiert, sondern statt dessen die Eingabefunktionen scheinbar einfrieren, so sollten sie folgende Zeilen in der Datei `$HOME/.dt/$LANG/dtwmrc` auskommentieren:

```
<Key>Down root f.circle_down
<Key>Up root f.circle_up
```

```
<Key>Right root f.next_workspace
```

```
<Key>Left root f.prev_workspace
```

In Tabelle 13.1 finden Sie Standardvariablen der Event-Klasse für verschiedene Tasten.

Tabelle 13.1: Standardtasten, die in der Event-Klasse definiert sind.

Klassenvariable	Taste
Event.HOME	(Pos_1)
Event.END	(Ende)
Event.PGUP	(Bild $\frac{1}{2}$)
Event.PGDN	(Bild $\frac{1}{4}$)
Event.UP	($\frac{1}{2}$)
Event.DOWN	($\frac{1}{4}$)
Event.LEFT	(æ)
Event.RIGHT	(Æ)
Event.F1	(F1)
Event.F2	(F2)
Event.F3	(F3)
Event.F4	(F4)
Event.F5	(F5)
Event.F6	(F6)
Event.F7	(F7)
Event.F8	(F8)
Event.F9	(F9)
Event.F10	(F10)
Event.F11	(F11)
Event.F12	(F12)

Beispiel: Zeichen eingeben, anzeigen und versetzen

Im folgenden wird ein Applet erläutert, das Tastaturereignisse im Ereignismodell 1.02 demonstriert. Bei diesem Applet kann man ein Zeichen auf der Tastatur eingeben, das dann in der Mitte des Applet-Fensters angezeigt wird. Anschließend läßt sich das Zeichen mit den Pfeiltasten am Bildschirm versetzen. Durch Eingabe eines weiteren Zeichens ändert sich das aktuell angezeigte Zeichen. Abbildung 13.3 zeigt ein Beispiel hierfür.



Abbildung 13.3:
Das Keys-Applet

Dieses Applet ist weniger komplex als die zuvor erläuterten Applets. Es verfügt über nur drei Methoden: `init()`, `keyDown()` und `paint()`. Die Instanzvariablen sind ebenfalls einfacher, weil Sie lediglich die x- und y-Positionen der aktuellen Zeichen und der Werte der betreffenden Zeichen verfolgen müssen. Zunächst wieder die Klassendefinition:

```
import java.awt.Graphics;
import java.awt.Event;
import java.awt.Font;
import java.awt.Color;

public class Keys extends java.applet.Applet {

    char currkey;
    int currx;
    int curry;
}
```

Die `init()`-Methode ist für drei Aufgaben zuständig: Definition der Hintergrundfarbe, Einrichten der Applet-Schriftarten (in diesem Fall 36 Punkt Helvetica fett) und Festlegung der Anfangsposition für das Zeichen (Bildschirmmitte abzüglich einiger Punkte für ein leichtes Versetzen des Zeichens nach rechts oben).

```
public void init() {
    currx = (size().width / 2) - 8;
    curry = (size().height / 2) - 16;
    setBackground(Color.white);
    setFont(new Font("Helvetica", Font.BOLD, 36));
    requestFocus();
}
```

Die letzte Anweisung in der `init()`-Methode erteilt dem Applet-Fenster den Eingabefokus. Diese Anweisung wird benötigt, um sicherzustellen, daß die Tastatureingaben auch von der Komponente empfangen werden, die diese behandeln sollen - hier das Applet-Fenster selbst.



In den früheren Versionen von Java war der Aufruf von `requestFocus()` nicht nötig, damit das Applet-Fenster Tastatureingaben empfangen konnte. Außerdem konnten Sie dem Applet-Fenster den Fokus

erteilen, indem Sie in dieses geklickt haben. Dies gilt auch für die aktuellsten Versionen des Netscape Navigator und des Microsoft Internet Explorer. Der Appletviewer des JDK 1.2 macht allerdings die Verwendung von `requestFocus()` erforderlich. Andernfalls erhält das Applet-Fenster den Eingabefokus für Tastatureingaben nicht. Behalten Sie diesen Unterschied im Gedächtnis, wenn Sie Applets testen, die Tastaturereignisse verwenden. Über die Methode `requestFocus()` dem Applet-Fenster den Fokus zu erteilen, ist wahrscheinlich der beste Weg.

Da das Verhalten des Applets auf Tastatureingaben basiert, findet die Hauptarbeit des Applets in der `keyDown()`-Methode statt:

```
public boolean keyDown(Event evt, int key) {
    switch (key) {
        case Event.DOWN:
            curry += 5;
            break;
        case Event.UP:
            curry -= 5;
            break;
        case Event.LEFT:
            currx -= 5;
            break;
        case Event.RIGHT:
            currx += 5;
            break;
        default:
            currkey = (char)key;
    }
    repaint();
    return true;
}
```

In der Mitte des `keyDown()`-Applets befindet sich eine `switch`-Anweisung, die auf verschiedene Tastenereignisse testet. Ist das Ereignis eine Pfeiltaste, wird die Position des Zeichens entsprechend geändert. Ist das Ereignis eine andere Taste, ändert sich das Zeichen selbst (dies ist der Standardteil von `switch`). Die Methode endet mit `repaint()` und gibt `true` aus.

Die `paint()`-Methode ist hier fast inhaltslos. Es wird lediglich das aktuelle Zeichen an der aktuellen Position angezeigt. Beachten Sie aber, daß es beim ersten Starten des Applets kein Anfangszeichen gibt und nichts zu zeichnen ist. Das muß berücksichtigt werden. Die Variable `currkey` wird mit 0 initialisiert, so daß das Applet nur gezeichnet wird, wenn `currkey` einen tatsächlichen Wert hat:

```
public void paint(Graphics g) {
    if (currkey != 0) {
        g.drawString(String.valueOf(currkey), currx, curry);
    }
}
```

Listing 13.3 zeigt den kompletten Quellcode für das Keys-Applet.

Listing 13.3: Der gesamte Quelltext von Keys.java

```
1: import java.awt.Graphics;
2: import java.awt.Event;
3: import java.awt.Font;
4: import java.awt.Color;
```

```
5:
6: public class Keys extends java.applet.Applet {
7:
8:     char currkey;
9:     int currx;
10:    int curry;
11:
12:    public void init() {
13:        currx = (size().width / 2) -8;  // Standard
14:        curry = (size().height / 2) -16;
15:
16:        setBackground(Color.white);
17:        setFont(new Font("Helvetica",Font.BOLD,36));
18:        requestFocus();
19:    }
20:
21:    public boolean keyDown(Event evt, int key) {
22:        switch (key) {
23:            case Event.DOWN:
24:                curry += 5;
25:                break;
26:            case Event.UP:
27:                curry -= 5;
28:                break;
29:            case Event.LEFT:
30:                currx -= 5;
31:                break;
32:            case Event.RIGHT:
33:                currx += 5;
34:                break;
35:            default:
36:                currkey = (char)key;
37:        }
38:
39:        repaint();
40:        return true;
41:    }
42:
43:    public void paint(Graphics g) {
44:        if (currkey != 0) {
45:            g.drawString(String.valueOf(currkey), currx,curry);
46:        }
47:    }
48: }
```

Sie können das Applet mit der folgenden HTML-Anweisung testen:

```
<applet code="Keys.class" height=100 width=100>
</applet>
```

Modifizierungstasten und Maustastenkombinationen

Zu den sogenannten Modifizierungstasten zählen die Umschalttaste (^a_), Steuerungstaste (Strg) und Meta. Sie erzeugen selbst keine Tastenereignisse, aber wenn Sie ein gewöhnliches Maus- oder Tastaturereignis erhalten, können Sie testen, ob eine dieser Tasten gedrückt wurde, während das Ereignis stattfand. In manchen Fällen ist das offensichtlich. Alphanumerische Tasten, die gleichzeitig mit der (^a_)-Taste gedrückt werden, erzeugen z.B. andere Zeichen als ohne. In anderen Fällen, insbesondere zusammen mit der Maus, soll ein Ereignis mit einer gedrückten Modifizierungstaste aktiviert werden, um es von der üblichen Version des jeweiligen Ereignisses zu unterscheiden.



Die Taste Meta wird meist bei Unix-Systemen verwendet und entspricht der Taste (Alt) auf PC-Tastaturen und Command ((°)-Taste) auf Macintosh.

Die Event-Klasse enthält drei Methoden, um zu testen, ob eine Taste gleichzeitig mit einer Ergänzungstaste gedrückt wurde: `shiftDown()`, `metaDown()` und `controlDown()`. Alle drei geben boolesche Werte zurück, die Auskunft darüber geben, ob die jeweilige Modifizierungstaste gedrückt wurde. Sie können diese drei Methoden in jeder beliebigen Ereignisbehandlung (Maus oder Tastatur) verwenden, indem Sie diese über das Ereignisobjekt aufrufen, das an die jeweilige Methode weitergegeben wurde.

```
public boolean mouseDown(Event evt, int x, int y ) {
    if (evt.shiftDown())
        // Shift-Klick verarbeiten
    else if controlDown()
        /// Strg-Klick verarbeiten
    else // Normalen Klick verarbeiten
}
```

Ein weiterer wichtiger Verwendungszweck der Modifizierungstasten-Methoden besteht darin, zu testen, welche Maustaste ein spezielles Mausereignis erzeugt hat - dies gilt für Systeme mit zwei oder drei Maustasten. Standardmäßig werden Mausereignisse (wie das Drücken und Ziehen) im Ereignismodell 1.02 unabhängig davon erzeugt, welche Maustaste benutzt wurde. Doch die Java-Ereignisse zeichnen interne Aktionen der linken oder mittleren Maustaste zusammen mit den Ergänzungstasten Meta und Steuerung (Strg) auf, d.h. ein Test auf die Tasten prüft die Maustastenaktion. Indem Sie die Modifizierungstasten testen, können Sie ermitteln, welche Maustaste gedrückt wurde, und für die jeweiligen Maustasten verschiedene Verhaltensweisen definieren. Hiermit läßt sich also nicht nur die linke Maustaste mit Ereignissen versehen. Verwenden Sie für diesen Test eine if-Anweisung wie folgt:

```
public boolean mouseDown(Event evt, int x, int y ) {
    if (evt.metaDown())
        // einen Klick mit der rechten Maustaste verarbeiten
    else if (evt.controlDown())
        // einen Klick mit der mittleren Maustaste verarbeiten
    else // einen normalen Klick verarbeiten
}
```

Da diese Aufzeichnung mehrerer Maustasten in Modifizierungstasten automatisch vorgenommen wird, müssen Sie nicht viel tun, um sicherzustellen, daß ein Applet oder eine Anwendung auch auf anderen Systemen mit anderen Mäusen funktioniert. Da die linken oder rechten Mausklicks in Modifizierungstasten-Ereignissen aufgezeichnet werden, können Sie die tatsächlichen Modifizierungstasten

auf einem System mit wenig Maustasten für exakt dieselben Ereignisse verwenden. Zum Beispiel: Das Drücken der Steuerungstaste und gleichzeitig Klicken mit der Maus unter Windows oder das Drücken der (Ctrl)-Taste auf einem Macintosh entspricht genau dem Klicken der mittleren Maustaste auf einer Maus mit drei Tasten. Wenn Sie die ($\frac{3}{4}$)-Taste (Apple) drücken und mit der Maustaste bei einem Macintosh klicken, ist diese identisch mit dem Klicken der rechten Maustaste bei einer Maus mit zwei oder drei Tasten.

Bedenken Sie jedoch, daß die Verwendung von verschiedenen Maustasten oder Modifizierungstasten eventuell nicht sofort deutlich wird, wenn das Applet oder die Anwendung auf einem System ausgeführt wird, das weniger Tasten zur Verfügung stellt, als bei Ihnen üblich. Sie können entweder die Oberfläche auf die Verwendung einer einzigen Maustaste beschränken oder in Form einer Hilfe oder Dokumentation darlegen, wie Ihr Programm in diesem Fall benutzt werden soll.

Der generische Eventhandler

Die Standardmethoden, die Sie heute zur Behandlung von grundlegenden Ereignissen kennengelernt haben, werden von einem generischen Event-Handler aufgerufen, einer Methode mit dem Namen `handleEvent()`. Die `handleEvent()`-Methode spiegelt die Art wider, in der das AWT 1.02 im allgemeinen mit Ereignissen umgeht, die zwischen Anwendungskomponenten und Ereignissen aufgrund von Benutzereingaben stattfinden.

In der Standardmethode `handleEvent()` werden Ereignisse verarbeitet und die Methoden, die Sie heute gelernt haben, werden aufgerufen. Um die hier beschriebenen Standardereignisse zu ändern oder eigene Ereignisse zu definieren und weiterzugeben, müssen Sie `handleEvent()` in Ihrem Java-Programm überschreiben. Die `handleEvent()`-Methode sieht wie folgt aus:

```
public boolean handleEvent(Event evt) {
    ...
}
```

Um spezielle Ereignisse zu testen, prüfen Sie die `id`-Instanzvariable des Event-Objekts, das darin weitergegeben wird. Die Ereignis-ID ist eine Ganzzahl, jedoch definiert die Event-Klasse glücklicherweise eine ganze Reihe von Ereignis-IDs als Klassenvariablen, die Sie im Rumpf von `handleEvent()` testen können. Da diese Klassenvariablen ganzzahlige Konstanten sind, eignet sich eine `switch`-Anweisung besonders gut. Die folgende `handleEvent()`-Methode ist ein einfaches Beispiel, bei dem Informationen über Mausereignisse ausgegeben werden:

```
public boolean handleEvent(Event evt) {
    switch (evt.id) {
        case Event.MOUSE_DOWN:
            System.out.println("MouseDown: " +
                               evt.x + ", " + evt.y);
            return true;
        case Event.MOUSE_UP:
            System.out.println("MouseUp: " +
                               evt.x + ", " + evt.y);
            return true;
        case Event.MOUSE_MOVE:
            System.out.println("MouseMove: " +
                               evt.x + ", " + evt.y);
            return true;
        case Event.MOUSE_DRAG:
            System.out.println("MouseDown: " +
                               evt.x + ", " + evt.y);
    }
```

```

        return true;
    default:
        return false;
    }
}

```

Sie können auf die folgenden Tastaturereignisse testen:

- Event.KEY_PRESS wird erzeugt, wenn der Benutzer eine Taste drückt (entspricht der keyDown()-Methode).
- Event.KEY_RELEASE wird erzeugt, wenn der Benutzer eine Taste losläßt.
- Event.KEY_ACTION und Event.KEY_ACTION_RELEASE werden erzeugt, wenn eine Aktions-Taste (Funktionstaste, (½) (¼) (Æ) (æ)-taste, (Bild½), oder (Bild¼) (Pos_1)) gedrückt wird.

Ferner können Sie auf folgende Mausereignisse testen:

- Event.MOUSE_DOWN wird erzeugt, wenn der Benutzer die Maustaste drückt (wie die mouseDown()-Methode).
- Event.MOUSE_UP wird erzeugt, wenn die Maustaste losgelassen wird (ebenso wie die mouseUp()-Methode).
- Event.MOUSE_MOVE wird erzeugt, wenn die Maus bewegt wird (wie die mouseMove()-Methode).
- Event.MOUSE_DRAG wird erzeugt, wenn die Maus mit gedrückter Maustaste bewegt wird (wie die mouseDrag()-Methode).
- Event.MOUSE_ENTER wird erzeugt, wenn die Maus den Applet-Bereich (oder eine Komponente davon) betritt.
- Event.MOUSE_EXIT wird erzeugt, wenn die Maus den Applet-Bereich verläßt. Sie können auch die mouseExit()-Methode verwenden.

Abgesehen von diesen Ereignissen enthält die Event-Klasse eine ganze Reihe von Methoden zur Behandlung von 1.02-AWT-Komponenten der Benutzeroberfläche. Sie lernen diese Methode morgen.

Beachten Sie ferner, daß handleEvent() ebenso wie die individuellen Methoden für die einzelnen Ereignisse einen booleschen Wert ausgibt. Der hier zurückgegebene Wert ist besonders wichtig; wenn Sie die Ereignisbehandlung an eine andere Methode weitergeben, müssen Sie false zurückgeben (die aufgerufene Methode selbst wird true oder false zurückgeben). Wenn die Behandlung des Ereignisses im Rumpf dieser Methode ausgeführt wird, geben Sie true zurück. Wird das Ereignis an eine Superklasse weitergeleitet, gibt diese Methode automatisch true oder false zurück, d.h. in diesem Fall müssen Sie dies nicht selbst zurückgeben.

Handhabung von Ereignissen der Benutzeroberfläche

Falls Sie mit dem Bearbeiten der heutigen Lektion an dieser Stelle aufgehört haben, verfügen Sie über das Wissen, um ein Applet zu erstellen, das viele kleine Komponenten der Benutzeroberfläche enthält, mit dem entsprechenden Layout-Manager ansprechend am Bildschirm ausgelegt ist sowie Abstände und Eckeinsätze hat. Allerdings wäre dann Ihr Applet ziemlich fade, weil die Komponenten der Benutzeroberfläche eigentlich nichts tun, wenn man sie anklickt oder die entsprechende Taste drückt.

Damit Ihre Komponenten der Benutzeroberfläche etwas bewirken, ist die Verknüpfung mit einem Code zur Handhabung von Ereignissen erforderlich. Denken Sie an die Maus- und Tastaturereignisse - genauso

werden durch Ereignisse der Benutzeroberfläche Reaktionen Ihrer Applets oder auf Eingaben durch den Benutzer hervorgerufen. Allerdings spielen sich die Ereignisse für AWT-Komponenten auf einer höheren Ebene ab; Schaltflächen beispielsweise verwenden Aktionsereignisse, die durch Anklicken der Schaltfläche ausgelöst werden. Sie brauchen sich über `mouseDown` oder `mouseUp` keine Gedanken zu machen; die Komponente übernimmt diese Aufgabe für Sie.

Konzentrieren wir uns heute auf die von den sechs grundlegenden Komponenten der Benutzeroberfläche erzeugten Ereignisse, mit denen Sie sich bereits beschäftigt haben. Eigentlich erzeugen lediglich fünf von ihnen Ereignisse (Labels sind statisch). Diese sechs Komponenten der Benutzeroberfläche können fünf Ereignisarten erzeugen:

- **Aktionsereignisse:** Das bei den meisten Komponenten der Benutzeroberfläche am Anfang stehende Ereignis zur Anzeige, daß diese Komponente »aktiviert« wurde. Aktionsereignisse werden durch Anklicken einer Schaltfläche erzeugt, durch Selektion oder Deselektion eines Kontroll- oder Optionsfeldes, durch Auswahl eines Elements aus einem Menü oder wenn der Benutzer die (€)- oder (Enter)- Taste in einem Textfeld drückt.
- **Ereignisse *Listenelement ausgewählt* und *Listenelement abgewählt*:** Diese Ereignisse werden erzeugt, wenn ein Kontrollfeld oder ein Element eines Listenfeldes gewählt wird (erzeugt ebenfalls ein Aktionsereignis).
- **Ereignisse bei Erhalt des Fokus oder Verlust des Fokus:** Diese Ereignisse können von jeder Komponente, entweder als Reaktion auf einen Mausklick oder als Teil eines Fokuswechsels durch Verwendung der (ÿ_)-Taste erzeugt werden. Bei Fokuserhalt bedeutet einfach, daß der Komponente der Eingabefokus zugewiesen wurde und nun entsprechende Aktivitäten durch Anklicken oder Drücken einer Taste ausgeführt werden können. Bei Fokusverlust hingegen bedeutet, daß der Eingabefokus einer anderen Komponente zugewiesen wurde.

Behandlung von Aktionsereignissen

Aktionsereignisse sind die bei weitem am häufigsten eingesetzten Ereignisse der Benutzeroberfläche, und daher wird für ihre Handhabung eine spezielle Methode angewandt, genau wie bei den grundlegenden Maus- und Tastaturereignismethoden, die wir gestern durchgenommen haben.

Um ein von einer beliebigen Komponente der Benutzeroberfläche erzeugtes Aktionsereignis abzufangen, definieren Sie in Ihrem Applet oder Klasse eine `action()`-Methode mit folgender Signatur:

```
public boolean action(Event evt, Object arg) {
    ...
}
```

Diese `action()`-Methode dürfte angesichts der bereits gelernten grundlegenden Ereignismethoden für Maus und Tastatur vertraut aussehen. Wie jene Methoden wird sie an das Event-Objekt, das die jeweilige Aktion darstellt, weitergegeben. Außerdem erhält sie ein zusätzliches Objekt (bei diesem Code den Parameter `arg`), das ein beliebiger Klassentyp sein kann.

Das zweite Argument für die `action()`-Methode hängt von der Komponente der Benutzeroberfläche ab, durch die die Aktion erzeugt wird. Die Basisdefinition ist ein durch die Komponente der Benutzeroberfläche bestimmtes »beliebiges arbiträres Argument« zur Weitergabe zusätzlicher Informationen, die Ihnen später bei der Bearbeitung der Aktion nützlich sein können. Tabelle 13.2 zeigt die zusätzlichen Argumente für jede Komponente der Benutzeroberfläche.

Tabelle 13.2: Aktionsargumente für jede Komponente der Benutzeroberfläche

Komponente	Argumenttyp	Enthält

Schaltfläche	String	Das Label der Schaltfläche
Kontrollfelder	Boolesch	Immer <code>true</code>
Optionsfelder	Boolesch	Immer <code>true</code>
Auswahlmenüs	Zeichenkette	Das Label des gewählten Elements
Textfelder	Zeichenkette	Textinhalt des Felds

Als erstes müssen Sie innerhalb der `action()`-Methode testen, welche Komponente der Benutzeroberfläche die Aktion erzeugt hat (im Gegensatz zu Ereignissen der Maus oder Tastatur, wo es nicht so eine große Rolle spielt, da alle unterschiedlichen Komponenten Aktionen erzeugen können). Um dies zu vereinfachen, beinhaltet das `Event`-Objekt, das Sie beim Aufruf von `action()` erhalten, eine `target`-Instanzvariable, die eine Referenz zu dem Objekt, das das Ereignis aufgenommen hat, enthält. Sie können den `instanceof`-Operator wie folgt benutzen, um herauszufinden, von welcher Komponente das Ereignis ausging:

```
public boolean action(Event evt, Object arg) {
    if (evt.target instanceof TextField)
        return handleText(evt.target);
    else if (evt.target instanceof Choice)
        return handleChoice(arg);
    //...

    return false;
}
```

In diesem Beispiel könnte `action()` entweder durch ein `TextField` oder ein `Auswahlmenü` erzeugt worden sein; die `if`-Anweisungen bestimmen, von welchem der beiden das Ereignis erzeugt wurde, und rufen zur entsprechenden Bearbeitung eine andere Methode auf (`handleText()` oder hier `handleChoice()`). (Weder `handleText()` noch `handleChoice()` sind AWT-Methoden. Es wurden einfach zwei beliebige Hilfsmethoden ausgewählt. Die Erzeugung dieser Hilfsmethoden ist allgemein üblich, um zu verhindern, daß `action()` mit viel Code vollgestopft wird.)

Wie bei anderen Ereignismethoden, gibt `action()` einen booleschen Wert aus, und Sie sollten `true` ausgeben, falls `action()` das Ereignis selbst behandelt, oder `false`, falls das Ereignis weitergegeben (oder ignoriert) wird. In unserem Beispiel haben Sie die Kontrolle an die Methode `handleText()` oder `handleChoice()` weitergegeben, und es ist deren Aufgabe, `true` oder `false` auszugeben, Sie können also `false` ausgeben (Sie erinnern sich, daß Sie `true` nur dann ausgeben, wenn diese Methode das Ereignis verarbeitet hat).

Zusätzliche Komplikationen können dann auftreten, wenn viele Komponenten mit denselben Klassen vorkommen - beispielsweise eine große Anzahl von Schaltflächen. Sie können alle Aktionen erzeugen und sind alle Instanzen von `Button`. Hier brauchen wir das zusätzliche Argument: Sie können die Labels, Elemente oder den Inhalt der Komponenten zur Bestimmung der das Ereignis erzeugenden Komponente und einfache Zeichenkettenvergleiche zur ihrer Auswahl verwenden. (Vergessen Sie dabei nicht, das Argument in den richtigen Objekt-Typ zu casten.)

```
public boolean action(Event evt, Object arg) {
    if (evt.target instanceof Button) {
        String labl = (String)arg;
        if (labl.equals("OK"))
            // OK-Schaltfläche handhaben
        else if (labl.equals("Cancel"))
            // Cancel-Schaltfläche handhaben
        else if (labl.equals("Browse"))
```

```
// Browse-Schaltfläche handhaben
```

```
...
```

```
}
```



Und wie ist das bei Kontroll- und Optionsfeldern? Da ihr zusätzliches Argument immer true ist, würde entsprechendes Testen nicht viel nützen. Im allgemeinen sollten Sie auf ein gewähltes Kontroll- oder Optionsfeld nicht reagieren. Normalerweise können Kontroll- und Optionsfelder frei vom Benutzer gewählt oder nicht gewählt werden, und ihre Werte werden dann an anderer Stelle überprüft (beispielsweise beim Anklicken einer Schaltfläche).

Um eine Reaktion Ihres Programms auf die Auswahl eines Kontroll- oder Optionsfelds zu initiieren, können Sie, anstatt das zusätzliche Argument zu verwenden, die Methode `getLabel()` benutzen, um das Label des Kontrollfelds im Rumpf der `action()`-Methode zu ermitteln. (Praktisch alle Komponenten verfügen über ähnliche Methoden; die Anwendung gestaltet sich nur leichter, wenn die Information als zusätzliches Argument übergeben wird.)

Im Abschnitt »Beispiel: Hintergrundfarbwechsler« erstellen Sie ein einfaches AWT-basiertes Applet, das Ihnen die Verwendung der `action()`-Methode in einer richtigen Anwendung zeigt.

Fokusereignisse behandeln

Wie bereits erwähnt, stellen Aktionsereignisse die bei weitem am häufigsten vorkommenden Ereignisse der Benutzeroberfläche dar, mit denen Sie sich im Hinblick auf die Komponenten, die in dieser Lektion behandelt wurden, beschäftigen werden. Allerdings können Sie vier weitere Ereignisarten in Ihrem eigenen Programm benutzen: Liste gewählt, Liste nicht gewählt, Fokuserhalt und Fokusverlust.

Für die Ereignisse Fokuserhalt und Fokusverlust können Sie die Methoden `gotFocus()` und `lostFocus()` verwenden, die in gleicher Weise wie `action()` eingesetzt werden. Hier ihre Signatur:

```
public boolean gotFocus(Event evt, Object arg) {
```

```
...
```

```
}
```

```
public boolean lostFocus(Event evt, Object arg) {
```

```
...
```

```
}
```

Für die Ereignisse Listeneintrag gewählt und Listeneintrag abgewählt stehen keine Methoden zur Verfügung, die Sie einfach überschreiben können. Für diese Ereignisse müssen Sie `handleEvent()` wie folgt verwenden:

```
public boolean handleEvent(Event evt) {
    if (evt.id == Event.LIST_SELECT)
        handleSelect(Event);
    else if (evt.id == Event.LIST_DESELECT)
        handleDeselect(Event);
    else return super.handleEvent(evt);
}
```

In diesem Codebruchstück sind `Event.LIST_SELECT` und `Event.LIST_DESELECT` die offiziellen

Ereigniskennzeichen für die Ereignisse Liste gewählt und Liste nicht gewählt. Hier wurde die Kontrolle einfach an die zwei Handler-Methoden `handleSelect()` und `handleDeselect()` weitergegeben, die theoretisch an anderer Stelle definiert sind. Beachten Sie den Aufruf an `super.handleEvent()` in der unteren Zeile; dieser Aufruf sorgt dafür, daß andere Ereignisse elegant in der Hierarchie an die ursprüngliche `handleEvent()`-Methode weitergegeben werden.

Ereignisse von mehrzeiligen Textfeldern

Mehrzeilige Textfelder erzeugen dieselben Ereignisse wie Textfelder. Sie können die Methoden `gotFocus()` und `lostFocus()` verwenden, um Fokusereignisse zu verarbeiten:

```
public boolean gotFocus(Event evt, Object arg) {
    // ...
}

public boolean lostFocus(Event evt, Object arg) {
    // ...
}
```

Ereignisse von Listenfeldern

Listenfelder erzeugen drei verschiedene Ereignisarten: Die Auswahl bzw. Abwahl eines Eintrags in der Liste erzeugen entsprechende Ereignisse bzw. ein Doppelklick auf einen Eintrag erzeugt ein Aktionsereignis.

Sie können die Methode `action()` überschreiben, um einen Doppelklick auf einen Eintrag in einer Liste zu behandeln. Die Aus- bzw. Abwahl eines Eintrags in einer Liste behandeln Sie, indem Sie die Methode `handleEvent()` überschreiben und auf die Ereignis-ID `LIST_SELECT` und `LIST_DESELECT` prüfen.

Ereignisse von Bildlaufleisten

Wenn Sie den Umgang mit Ereignissen mögen, dann werden Sie Bildlaufleisten lieben. Eine ganze Reihe von Ereignissen wird allein von den verschiedenen Bewegungen innerhalb einer Bildlaufleiste erzeugt. Sie müssen die Methode `handleEvent()` für alle diese Ereignisse verwenden. In Tabelle 13.3 sind die Ereignis-IDs aufgeführt, auf die Sie prüfen müssen, und die Bewegung, die die Ereignisse auslöst.

Tabelle 13.3: Ereignisse von Bildlaufleisten

Ereignis-ID	Beschreibung
SCROLL_ABSOLUTE	Wird erzeugt, wenn der Schieber der Bildlaufleiste bewegt wird.
SCROLL_LINE_DOWN	Wird erzeugt, wenn die linke oder untere Schaltfläche der Bildlaufleiste angeklickt wird.
SCROLL_LINE_UP	Wird erzeugt, wenn die rechte oder obere Schaltfläche der Bildlaufleiste angeklickt wird.
SCROLL_PAGE_DOWN	Wird erzeugt, wenn der Bereich unter oder links von dem Schieber angeklickt wird.
SCROLL_PAGE_UP	Wird erzeugt, wenn der Bereich über oder rechts von dem Schieber angeklickt wird.

Beispiel: Hintergrundfarbwechsler

Aufgrund von Codebruchstücken allein die Zusammenhänge aller Teile zu verstehen, ist schwer. Machen wir uns also an die Lösung dieses Problems und erstellen ein einfaches AWT-Applet. Morgen, nachdem Sie mehr über die komplexeren Teile des AWT gelernt haben, werden wir uns mit einem etwas anspruchsvolleren Applet befassen, um den bisherigen Lerninhalt zu untermauern.

Das in Abbildung 13.4 dargestellte Applet, das Sie in diesem Abschnitt erstellen werden, verwendet fünf Schaltflächen, die übersichtlich oben am Bildschirm angeordnet sind, wobei jede mit dem Namen einer Farbe beschriftet ist. Jede Schaltfläche ändert die Hintergrundfarbe des Applets in die auf der Schaltfläche vermerkte Farbe.



Abbildung 13.4:
Das Applet SetBack

Der erste Schritt in diesem Abschnitt besteht allerdings darin, den Code der Benutzeroberfläche zu erzeugen. Im allgemeinen ist dies der beste Weg zur Erstellung eines AWT-basierten Applets: Erzeugen Sie die Komponenten und das Layout, und stellen Sie sicher, daß alles richtig aussieht, bevor Sie sich an die Einbindung der Ereignisse und damit an die eigentliche Arbeit mit dem Applet machen.

Bei diesem Applet sind die Komponenten und das Layout ausgesprochen einfach gehalten. Das Applet beinhaltet fünf einfache, oben am Bildschirm in einer Reihe angeordnete Schaltflächen. Ein `FlowLayout` eignet sich für diese Anordnung am besten und erfordert wenig Arbeit.

Hier der Code der für dieses Applet erzeugten Klassenstruktur und `init()`-Methode. Das `FlowLayout` ist zentriert, und jede Schaltfläche hat einen Abstand von 10 Punkten. Anschließend müssen Sie lediglich die einzelnen Schaltflächen erstellen und hinzufügen.

```
import java.awt.*;
```

```
public class SetBack extends java.applet.Applet {
```

```
    Button redButton, blueButton, greenButton,  
        whiteButton, blackButton;
```



```

public void init() {
    setBackground(Color.white);
    setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));

    redButton = new Button("Red");
    add(redButton);
    blueButton = new Button("Blue");
    add(blueButton);
    greenButton = new Button("Green");
    add(greenButton);
    whiteButton = new Button("White");
    add(whiteButton);
    blackButton = new Button("Black");
    add(blackButton);
}

```

Auf den ersten Blick sieht dieser Code wahrscheinlich umfangreicher als notwendig aus. Sie könnten dagegen setzen, daß man zur Unterbringung der Schaltflächen eigentlich nicht alle Instanzvariablen benötigt. Das Ganze ist in der Tat ein bißchen undurchsichtig; da das Applet bereits erstellt ist, ist die geeignete Schreibweise bekannt, und die Instanzvariablen sorgen später für Erleichterung (haben Sie Vertrauen). Es wird häufiger vorkommen, daß, wenn Sie Ihre eigenen Applets schreiben, der von Ihnen ursprünglich für die Benutzeroberfläche geschriebene Code nicht gut funktioniert und Sie ihn entsprechend ändern müssen. Das macht überhaupt nichts! Je mehr Applets Sie schreiben, desto leichter werden Sie verstehen, wie letztendlich alles zusammenpaßt.

Ereigniscode einfügen

Das Anklicken von Schaltflächen löst Aktionsereignisse aus. Wie Sie bereits wissen, verwenden Sie zur Handhabung eines Aktionsereignisses die Methode `action()`. Diese `action()`-Methode wird hier folgendes auslösen:

- Sie testet, daß das Ziel des Ereignisses tatsächlich eine Schaltfläche ist.
- Sie testet weiter, welche Schaltfläche effektiv angeklickt wurde.
- Sie ändert den Hintergrund auf die durch die Schaltfläche gekennzeichnete Farbe.
- Sie ruft `repaint()` auf (nur die Aktion der Hintergrundänderung reicht nicht aus).

Bevor wir uns dem Schreiben der `action()`-Methode widmen, ist noch eine weitere designbezogene Entscheidung zu treffen. Im wesentlichen sind die letzten drei Schritte - bis auf kleine Unterschiede - für jede Schaltfläche identisch, so daß es wirklich Sinn macht, sie in einer eigenen Methode unterzubringen. Sie heißt `changeColor()` und wird zur Vereinfachung der Logik in `action()` beitragen.

Auf dieser Grundlage gestaltet sich die `action()`-Methode einfach:

```

public boolean action(Event evt, Object arg) {
    if (evt.target instanceof Button) {
        changeColor((Button)evt.target);
        return true;
    } else return false;
}

```

Diese `action()`-Methode unterscheidet sich wenig von den einfachen, im Abschnitt über Aktionen erzeugten. Der erste Schritt beinhaltet die Verwendung von `evt.target`, um sicherzustellen, daß die Komponente eine Schaltfläche ist. An dieser Stelle geben Sie die Kontrolle an die noch zu schreibende

Methode `changeColor()` weiter und geben `true` aus. Falls das Ereignis keine Schaltfläche ist, geben Sie `false` aus.

Beachten Sie das eine Argument für `changeColor()`. Mit diesem Argument geben Sie das eigentliche Schaltflächenobjekt, das das Ereignis erhalten hat, an die Methode `changeColor()` weiter. (Das Objekt in `evt.target` ist eine Instanz der Klasse `Object`; es muß also in einen `Button`-Objekt gecastet werden, damit Sie es als Schaltfläche verwenden können.) Ab hier wird die Methode `changeColor()` dies handhaben.

Apropos `changeColor()`, machen wir weiter und definieren jetzt diese Methode. Die Methode `changeColor()` ist hauptsächlich darauf ausgerichtet festzustellen, welche Schaltfläche angeklickt wurde. Sie erinnern sich, daß das zusätzliche Argument bei `action()` das Label der Schaltfläche war. Obwohl Sie mit einem Zeichenkettenvergleich in `changeColor()` herausfinden können, welche Schaltfläche angeklickt wurde, stellt das nicht die eleganteste Lösung dar und macht Ihren Ereigniscode in zu starkem Maße von der grafischen Benutzeroberfläche abhängig. Falls Sie ein Schaltflächen-Label ändern möchten, müssen Sie noch einmal zurückgehen und auch Ihren Ereigniscode neu bearbeiten. In diesem Applet können Sie somit das zusätzliche Argument vollkommen ignorieren.

Wie wissen Sie nun, welche Schaltfläche angeklickt wurde? An dieser Stelle kommen die Instanzvariablen der Schaltfläche ins Spiel. Das in der Instanzvariablen `target` des Ereignisses enthaltene Objekt - das Sie an `changeColor()` weitergegeben haben - ist eine Instanz von `Button`, und eine dieser Instanzvariablen enthält eine Referenz zu genau demselben Objekt. Sie müssen die beiden nur in `changeColor()` vergleichen, um zu sehen, ob sie dasselbe Objekt darstellen, den Hintergrund einstellen und neu zeichnen, und zwar folgendermaßen:

```
void changeColor(Button b) {
    if (b == redButton) setBackground(Color.red);
    else if (b == blueButton) setBackground(Color.blue);
    else if (b == greenButton) setBackground(Color.green);
    else if (b == whiteButton) setBackground(Color.white);
    else setBackground(Color.black);

    repaint();
}
```

Anklicken einer Schaltfläche von der Benutzeroberfläche aus ruft `action()` auf, `action()` ruft `changeColor()` auf und `changeColor()` stellt den entsprechenden Hintergrund ein. Ganz einfach! Listing 13.4 zeigt das fertige Applet.

Listing 13.4: Der gesamte Quelltext von `SetBack.java`

```
1: import java.awt.*;
2:
3: public class SetBack extends java.applet.Applet {
4:
5:     Button redButton, blueButton, greenButton, whiteButton, blackButton;
6:
7:     public void init() {
8:         setBackground(Color.white);
9:         setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));
10:
11:         redButton = new Button("Red");
12:         add(redButton);
13:         blueButton = new Button("Blue");
14:         add(blueButton);
15:         greenButton = new Button("Green");
```

```

16:         add(greenButton);
17:         whiteButton = new Button("White");
18:         add(whiteButton);
19:         blackButton = new Button("Black");
20:         add(blackButton);
21:     }
22:
23:     public boolean action(Event evt, Object arg) {
24:         if (evt.target instanceof Button) {
25:             changeColor((Button)evt.target);
26:             return true;
27:         } else return false;
28:     }
29:
30:     void changeColor(Button b) {
31:         if (b == redButton) setBackground(Color.red);
32:         else if (b == blueButton) setBackground(Color.blue);
33:         else if (b == greenButton) setBackground(Color.green);
34:         else if (b == whiteButton) setBackground(Color.white);
35:         else setBackground(Color.black);
36:
37:         repaint();
38:     }
39: }

```

Sie können dieses Applet testen, indem Sie den folgenden HTML-Code verwenden:

```

<applet code="SetBack.java" width=200 height=200>
</applet>

```

Zusammenfassung

Die Beendigung der Arbeit des heutigen Tages stellt in Ihrer Java-Programmierkarriere ein großes Ereignis dar. Die Fähigkeit, Ereignisse zu verarbeiten, macht es möglich, voll funktionsfähige Applets mit einer grafischen Benutzeroberfläche zu schreiben, die zur Interaktion mit dem Benutzer verwendet werden kann.

Morgen werden Sie Ihr Wissen um das Abstract Windowing Toolkit mit einem ausgefeilteren Projekt vertiefen. Außerdem werden Features, wie z.B. eigenständige Fenster, behandelt.

Während der dritten Woche erhalten Sie die Chance, eine funktionierende Applikation, die Swing - das neue Paket für grafische Benutzeroberflächen, das mit Java 1.2 eingeführt wurde - verwendet, zu erstellen.

Fragen und Antworten

Frage:

Ich habe eine neue Schaltflächenklasse, die ich im Aussehen unterschiedlich zu den Standard-AWT-Schaltflächenobjekten in 1.02 definiert habe. Ich möchte Aufrufe auf diese Schaltfläche rückführen (d.h. eine arbiträre Funktion bei Anklicken der Schaltfläche ausführen), ich kann aber nicht herausfinden, wie ich mit Java eine arbiträre Methode ausführen kann. In C++ verfüge ich nur über einen auf eine Funktion weisenden Zeiger. In Small-Talk würde ich perform verwenden: Wie kann ich das in Java durchführen?

Antwort:

In Java 1.02 können Sie es nicht; Aktionen der Schaltfläche werden durch ein action-Ereignis getriggert, das in derselben Klasse wie die Schaltfläche enthalten sein muß. Sie müssen Ihre Schaltflächenklasse jedesmal ableiten, wenn Sie für diese Schaltfläche ein anderes Verhalten erzeugen möchten. Dies ist einer der Gründe dafür, daß das Ereignismodell nach Java 1.02 geändert wurde; es ist wesentlich einfacher und effizienter, Ihre eigenen Komponenten zu erzeugen, wenn der Ereigniscode nicht zu stark an den Code der grafischen Benutzeroberfläche gebunden ist.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Tag 14

Fortgeschrittene Benutzeroberflächen mit dem AWT

Dies ist der letzte Tag, an dem Sie etwas über das Abstract Windowing Toolkit lernen. Ob Sie dies als gute oder schlechte Nachricht werten, hängt wahrscheinlich davon ab, wie vertraut der Umgang mit den Klassen des AWT für Sie geworden ist.

Wenn Sie glauben, daß dies eine gute Nachricht ist, dann werden Sie sich in bezug auf das AWT besser fühlen, wenn Sie heute einige seiner fortgeschritteneren Features kennengelernt haben.

Sie bauen auf allem, was Sie an den vorangegangenen Tagen über Komponenten, Layout-Manager und Ereignisse der Benutzerschnittstelle gelernt haben, auf und erhalten eine Einführung in einige neue Konzepte:

- So funktionieren Komponenten und die verschiedenen Dinge, die Sie mit diesen tun können.
- Fenster, Frames und Dialogfelder
- Menüs
- Eigenständige AWT-Applikationen erzeugen

Fenster, Frames und Dialogfelder

Zusätzlich zu den Grafiken, Ereignissen, Komponenten der Benutzeroberfläche und Layoutmechanismen bietet AWT Möglichkeiten zur Erstellung von Elementen der Benutzeroberfläche außerhalb eines Applets oder Browsers: Fenster, Menüs und Dialogfelder. Damit können Sie ausgewachsene Anwendungen als Teil Ihres Applets oder als unabhängige Java-Applikationen erstellen.

Die AWT-Fensterklassen

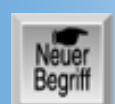
Die Java-AWT-Klassen zur Erstellung von Fenstern und Dialogfeldern erben von einer einzigen Klasse: Window. Die Window-Klasse, die selbst wiederum von Container erbt (und daher eine standardmäßige AWT-Komponente ist), liefert das allgemeine Verhalten für alle fensterähnlichen Elemente. Im allgemeinen verwenden Sie keine Instanzen von Window, sondern setzen Instanzen von Frame oder Dialog ein.

Die Frame-Klasse bietet ein Fenster mit einer Titelleiste, einer Schließen-Schaltfläche und anderen, plattformabhängigen Fenstermerkmalen. In diesen Frames können Sie auch Menüleisten einfügen. Dialog hingegen ist eine eingeschränkere Form von Frame, die normalerweise nicht über einen Titel verfügt. FileDialog, eine Subklasse von Dialog, bietet ein Standard-Dateiauswahldialogfeld (das normalerweise wegen der Sicherheitsbeschränkungen nur innerhalb von Java-Applikationen genutzt werden kann).

Wenn Sie ein neues Fenster oder Dialogfeld in Ihr Applet oder Ihre Applikation aufnehmen wollen, erstellen Sie Subklassen zu den Klassen Frame und Dialog.

Frames

Frames sind Fenster, die von dem Applet oder Browser unabhängig sind, der das Applet enthält. Es handelt sich hierbei um eigenständige Fenster mit eigenen Titeln, Schaltflächen zum Verkleinern, Vergrößern und Schließen sowie Menüleisten. Sie können Frames für Ihre eigenen Applets erstellen, um Fenster zu erzeugen, oder Frames in Java-Applikationen nutzen, um den Inhalt dieser Applikation aufzunehmen.



Ein *Frame* ist ein plattformspezifisches Fenster mit einem Titel, einer Menüleiste, Schaltflächen zum Verkleinern, Vergrößern und Schließen sowie anderen Fensterfunktionen.

Um einen Frame zu erstellen, benutzen Sie einen der folgenden Konstruktoren:

- `new Frame()` erstellt einen Basisframe ohne Titel.
- `new Frame(String)` erstellt einen Basisframe mit dem angegebenen Titel.

Da Frame von Window abgeleitet ist, dieses wiederum von Container und jenes wiederum von Component, werden Frames grundsätzlich so wie andere AWT-Komponenten erstellt und eingesetzt. Rahmen sind Container wie Panels, so daß Sie andere Komponenten mit der `add()`-Methode einfügen können. Das Standardlayout für Frames ist `BorderLayout`. Hier ein Beispiel für die Erstellung eines Frames, das Festlegen des Layouts und Hinzufügen von zwei Schaltflächen:

```
win = new Frame("My Cool Window");
win.setLayout(new BorderLayout(10, 20));
win.add("North", new Button("Start"));
win.add("Center", new Button("Move"));
```

Um die Größe für einen neuen Frame festzulegen, verwenden Sie die `resize()`-Methode mit der Breite und Höhe des neuen Fensters. Diese Codezeile beispielsweise ändert die Fenstergröße auf 100 Pixel Breite und 200 Pixel Höhe:

```
win.resize(100, 200);
```

Beachten Sie jedoch folgendes: Da unterschiedliche Systeme eine unterschiedliche Vorstellung davon haben, was ein Pixel ist und diese Pixel in unterschiedlicher Auflösung darstellen, ist es schwierig, ein Fenster zu erstellen, das die »richtige« Größe bei jeder Plattform hat. Fenster, die auf einer Plattform in Ordnung sind, können auf einer anderen viel zu groß oder viel zu klein sein. Ein Ausweg aus diesem Problem ist, die Methode `pack()` anstatt `resize()` zu verwenden. Die `pack()`-Methode, die keine Argumente hat, erstellt ein Fenster in kleinstmöglicher Größe auf der Basis der aktuellen Größe der Komponenten im Fenster sowie des eingesetzten Layout-Managers und der eingesetzten Eckeinsätze. Im folgenden Beispiel werden zwei Schaltflächen erstellt und in ein Fenster eingefügt. Das Fenster wird dann auf die kleinstmögliche Größe reduziert, die diese Schaltflächen enthalten kann.

```
win = new Frame("My Other Cool Window");
win.setLayout(new FlowLayout());
win.add("North", new Button("OK"));
win.add("Center", new Button("Cancel"));
win.pack();
```

Ein neu erstelltes Fenster ist zunächst unsichtbar. Sie müssen die `show()`-Methode anwenden, um dieses Fenster am Bildschirm anzeigen zu lassen (und Sie können es mit `hide()` wieder verbergen):

```
win.show();
```

Wenn Sie Pop-up-Fenster aus einem Applet heraus aktivieren, kann der Browser auf irgendeine Art deutlich machen, daß das Fenster kein reguläres Browser-Fenster ist - normalerweise erscheint dann eine Warnung im Fenster selbst. In Netscape sagt eine gelbe Leiste unten in jedem Fenster *Untrusted Java Window*. Diese Warnung soll den Benutzer informieren, daß das Fenster aus einem Applet kommt und nicht vom Browser selbst. (Denken Sie daran, daß die *Frame*-Klasse Fenster erzeugt, die genauso aussehen wie ein normales Fenster.) Dies soll davor bewahren, ein schlechtes Applet zu erstellen, das den Benutzer beispielsweise zur Eingabe des Paßworts auffordert. Sie können nichts gegen diese Warnung unternehmen - sie bleibt immer, wenn Sie Fenster mit Applets zusammen benutzen.

Die Listings 14.1 und 14.2 zeigen die Klassen, die ein einfaches Applet mit einem Pop-up-Fenster erzeugen. Sowohl das Applet als auch das Fenster sind in Abbildung 14.1 dargestellt. Das Applet verfügt über zwei Schaltflächen: eine zum Anzeigen und eine zum Verbergen des Fensters. Der Fensterframe selbst, der aus einer Subklasse erstellt wird, die ich *BaseFrame1* genannt habe, enthält ein Label: *This is a Window*. Ich werde mich weiterhin auf dieses Fenster und dieses Applet beziehen, damit Sie besser verstehen, was passiert, und es später für Sie leichter wird.

Listing 14.1: Der gesamte Quelltext von *PopUpWindow.java*

```
1: import java.awt.*;
2:
3: public class PopUpWindow extends java.applet.Applet {
4:     Frame window;
5:     Button open, close;
6:
7:     public void init() {
```

```

8:         open = new Button("Open Window");
9:         add(open);
10:        close = new Button("Close Window");
11:        add(close);
12:
13:        window = new BaseFrame1("A Pop Up Window");
14:        window.resize(150,150);
15:    }
16:
17:    public boolean action(Event evt, Object arg) {
18:        if (evt.target instanceof Button) {
19:            String label = (String)arg;
20:            if (label.equals("Open Window")) {
21:                if (!window.isShowing())
22:                    window.show();
23:            } else {
24:                if (window.isShowing())
25:                    window.hide();
26:            }
27:            return true;
28:        } else
29:            return false;
30:    }
31: }

```

Listing 14.2: Der gesamte Quelltext von BaseFrame1.java

```

1: import java.awt.*;
2:
3: class BaseFrame1 extends Frame {
4:     String message = "This is a Window";
5:     Label l;
6:
7:     BaseFrame1(String title) {
8:         super(title);
9:         setLayout(new BorderLayout());
10:
11:         l = new Label(message, Label.CENTER);
12:         l.setFont(new Font("Helvetica", Font.PLAIN, 12));
13:         add("Center", l);
14:     }
15:
16:     public Insets getInsets() {
17:         return new Insets(20,0,25,0);
18:     }
19: }

```

Nachdem Sie beide Klassen kompiliert haben, kann das Applet mit dem folgenden HTML-Code getestet werden:

```

<applet code="PopUpWindow.class" height=200 width=200>
</applet>

```



Abbildung 14.1:
Fenster

Dieses Beispiel ist aus zwei Klassen gebildet: Die erste, `PopupWindow`, ist die Applet-Klasse, die das Pop-up-Fenster erzeugt und steuert. In der `init()`-Methode dieser Klasse, und hier insbesondere in den Zeilen 7 bis 15 im Listing 14.1, fügen Sie zwei Schaltflächen in das Applet ein, die das Fenster steuern. Dann wird das Fenster selbst erstellt, in seiner Größe verändert und angezeigt.

Die Steuerung in diesem Applet wird aktiv, wenn eine der Schaltflächen aktiviert wird. An dieser Stelle kommt die zweite Klasse ins Spiel. Die `action()`-Methode in den Zeilen 17-30 von Listing 14.1 verarbeitet die Klicks auf diese Schaltfläche, die Aktionsereignisse erzeugt. In dieser Methode wird mit der Schaltfläche `Open Window` einfach nur das Fenster angezeigt, wenn es verborgen ist (Zeilen 20 bis 22 in Listing 14.1), und es wird verborgen, wenn es angezeigt ist (Zeilen 23 bis 25).

Das Pop-up-Fenster selbst ist ein spezieller Frame mit Namen `BaseFrame1`. Bei diesem Beispiel ist der Frame recht einfach: er nutzt ein `BorderLayout` und zeigt ein Label in der Mitte des Frames. Beachten Sie, daß die Initialisierung des Frames in einem Konstruktor erfolgt, und nicht über eine `init()`-Methode. Da Frames normale Objekte und keine Applets sind, müssen Sie sie auf konventionelle Art und Weise initialisieren.

Im Konstruktor von `BaseFrame1` sehen Sie, daß die erste Zeile (Zeile 8) einen Aufruf des Konstruktors der übergeordneten Klasse zu `BaseFrame1` enthält. Wie Sie schon während Tag 6 gelernt haben, ist dieser Aufruf der erste Schritt bei der Initialisierung einer neuen Klasse. Vergessen Sie diesen Schritt nicht, wenn Sie Ihre eigenen Klassen erstellen - schließlich wissen Sie nie, welche wichtigen Dinge die Superklasse in dem Konstruktor macht.

Dialogfelder

Dialogfelder ähneln in ihrer Funktionsweise den Frames insoweit, daß ein neues Fenster am Bildschirm eingeblendet wird. Jedoch sind Dialogfelder als Übergangsfenster gedacht - z.B. Fenster, die Warnungen ausgeben, Sie nach bestimmten Informationen fragen etc. Dialoge haben für gewöhnlich keine Titelleisten und zeigen auch viele andere allgemeine Fenstermerkmale nicht (Sie können jedoch ein Dialogfeld mit einer Titelleiste erstellen). Dialogfelder können als nicht in ihrer Größe veränderbar oder auch modal angelegt werden. (Modale Dialogfelder verhindern die Eingabe in einem anderen, derzeit angezeigten Fenster, bis das Dialogfeld wieder geschlossen ist.)



Dialogfelder sind Übergangsfenster, die dazu dienen, den Benutzer über Ereignisse zu informieren oder Eingaben vom Benutzer anzufordern. Im Gegensatz zu Frames haben Dialogfelder im allgemeinen keine Titelleiste oder Schaltfläche zum Schließen des Felds.

Ein *modales Dialogfeld* verhindert die Eingabe in einem anderen, derzeit angezeigten Fenster, bis das Dialogfeld geschlossen ist. Ein modales Dialogfenster können Sie nicht auf Symbolgröße reduzieren, und es können parallel keine anderen Fenster in den Vordergrund geholt werden. Das modale Dialogfeld muß erst geschlossen sein, bevor Sie im System weiterarbeiten können. Typische Beispiele für modale Dialogfelder sind Warnungen und Alarme.

Das AWT bietet zwei Arten von Dialogfeldern: die Dialog-Klasse, die ein allgemeines Dialogfeld enthält, und `FileDialog`, womit ein plattformspezifischer Datei-Browser erzeugt wird.

Dialogobjekte

Dialoge werden fast genauso erstellt und eingesetzt wie Fenster. Um einen allgemeinen Dialog zu erstellen, benutzen Sie einen der folgenden Konstruktoren:

- `Dialog(Frame, boolean)` erzeugt einen unsichtbaren Dialog, der dem aktuellen Rahmen beigelegt und entweder modal (`true`) oder nichtmodal (`false`) ist.
- `Dialog(Frame, String, boolean)` erzeugt einen unsichtbaren Dialog mit dem angegebenen Titel entweder modal (`true`) oder nichtmodal (`false`).



Ein *Dialogfenster* ist, wie auch ein `Frame`, ein `Panel`, in dem Sie Komponenten der Benutzeroberfläche anordnen und zeichnen sowie Grafikoperationen ausführen können. Wie auch andere Fenster, ist das Dialogfeld zunächst unsichtbar. Sie können jedoch mit `show()` und `hide()` angezeigt bzw. wieder verborgen werden.

Fügen wir jetzt in das Beispiel mit dem Pop-up-Fenster ein Dialogfeld ein. Von den zwei Klassen in diesem Applet muß nur `BaseFrame1` geändert werden. Hier ändern Sie die Klasse dahingehend, daß sie eine Schaltfläche `Set Text` und eine neue Klasse, `TextDialog`, enthält, die ein Texteingabedialogfeld ähnlich dem in Abbildung 14.2 gezeigten enthält.

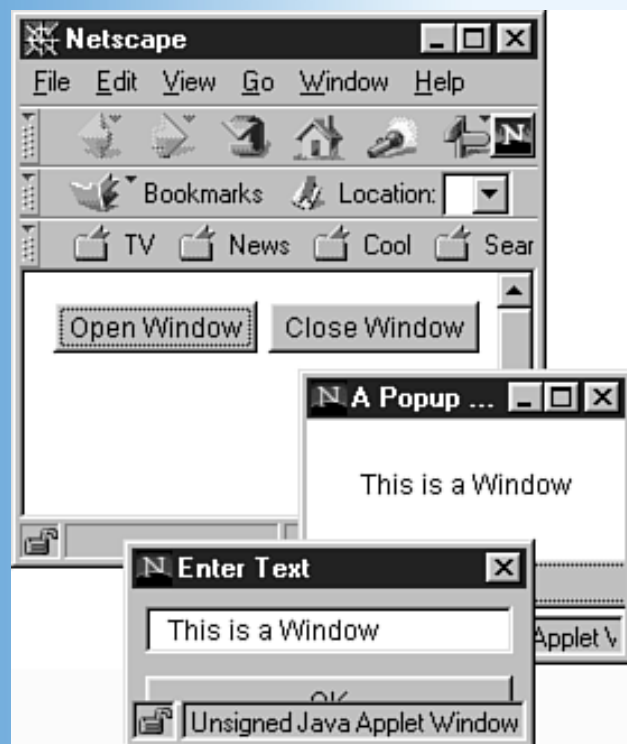


Abbildung 14.2:
Das Dialogfeld Enter Text



Im Codebeispiel auf der CD ist diese Version des Applets von der vorherigen Version getrennt. Ich habe eine neue Klasse namens `BaseFrame2` für diesen Teil des Beispiels erstellt, sowie eine neue Klasse `PopupWindowDialog.java`, die das Applet darstellt, zu dem dieses Fenster gehört. Außerdem habe ich `PopupActions2.java` als die Klasse erstellt, die die

Aktionen handhabt. Mit `PopupWindowDialog.html` kann diese Version des Applets angezeigt werden.

Um das Dialogfeld in die `BaseFrame1`-Klasse einzufügen, sind nur geringfügige Änderungen erforderlich. Als erstes ändern Sie den Namen der Klasse von `BaseFrame1` in `BaseFrame2`. Dann brauchen Sie eine Instanzvariable, die den Dialog enthält, weil Sie sich in der gesamten Klasse darauf beziehen werden:

```
TextDialog dl;
```

In der Konstruktor-Methode von `BaseFrame2` können Sie das Dialogfeld erzeugen (als Instanz der neuen Klasse `TextDialog`, die Sie in ein paar Minuten anlegen werden), es der Instanzvariablen `dl` zuweisen und in seiner Größe ändern (wie in den folgenden beiden Codezeilen gezeigt). Das Dialogfeld soll noch nicht angezeigt werden, weil es erst eingeblendet werden soll, wenn die Schaltfläche `Set Text` aktiviert wird.

```
dl = new TextDialog(this, "Enter Text", true);
dl.resize(150,100);
```

Als nächstes erstellen Sie die Schaltfläche `Set Text` in ähnlicher Form wie andere Schaltflächen, und dann fügen Sie sie in `BorderLayout` an der Position `"South"` ein (d.h. direkt unterhalb des Labels).

```
Button b = new Button("Set Text");
add("South", b);
```

Nachdem Sie ein `TextDialog`-Objekt und die `Set-Text`-Schaltfläche der Klasse `BaseFrame2` hinzugefügt haben, müssen Sie noch die folgende Methode zur Ereignisbehandlung einfügen:

```
public boolean action(Event evt, Object arg) {
    if (evt.target instanceof Button) {
        dl.show();
        return true;
    } else
        return false;
}
```

Dies zeigt das `TextDialog`-Objekt `dl` an, sobald irgendein `Button`-Objekt in dem Frame angeklickt wurde. In diesem Beispiel gibt es nur eine Schaltfläche bzw. `Button`-Objekt - `Set Text`.

```
window = new BaseFrame2("A Pop Up Window");
```

Das restliche Verhalten steckt in der `TextDialog`-Klasse, deren Code Sie in Listing 14.3 sehen.

Listing 14.3: Der gesamte Quelltext von `TextDialog.java`

```
1: import java.awt.*;
2:
3: class TextDialog extends Dialog {
4:     TextField tf;
5:     BaseFrame2 theFrame;
6:
7:     TextDialog(Frame parent, String title, boolean modal) {
8:         super(parent, title, modal);
9:
10:         theFrame = (BaseFrame2)parent;
11:         setLayout(new BorderLayout(10,10));
12:         setBackground(Color.white);
13:         tf = new TextField(theFrame.message,20);
14:         add("Center", tf);
15:
16:         Button b = new Button("OK");
17:         add("South", b);
18:     }
19:
20:     public Insets insets() {
21:         return new Insets(30,10,10,10);
22:     }
23:
```



```

24:     public boolean action(Event evt, Object arg) {
25:         if (evt.target instanceof Button) {
26:             String label = (String)arg;
27:             if (label == "OK") {
28:                 hide();
29:                 theFrame.l.setText(tf.getText());
30:             }
31:             return true;
32:         } else
33:             return false;
34:     }
35: }

```

Bei diesem Code sind ein paar Punkte zu beachten. Zunächst einmal achten Sie darauf, daß im Gegensatz zu den anderen beiden Fenstern in diesem Applet die Ereignisbehandlung innerhalb der Klasse stattfindet, so daß das Dialogfeld als seine eigene Ereignisbehandlung dient. Manchmal ist es sinnvoll, den Code Ereignisbehandlung separat zu schreiben, doch manchmal ist es einfacher, alles zusammenzulassen. In diesem Fall ist das Element `TextDialog` einfach genug, so daß es einfacher ist, alles zusammenzustellen.

Trotzdem sind in diesem Dialogfeld viele Elemente mit denen der `BaseFrame2`-Klasse identisch. Beachten Sie, daß der Konstruktor für `TextDialog` mit dem Konstruktor der Superklasse `Dialog` identisch ist, denn trotz der Tatsache, daß `TextDialog` einem Objekt verbunden ist, dessen Klasse `BaseFrame2` ist, müssen die Dialogfelder einem `Frame`- Objekt zugeordnet werden. Es ist leichter, den Konstruktor allgemeiner zu erstellen und ihn anschließend zu spezialisieren, nachdem der Konstruktor der Superklasse aufgerufen wurde - und genau das geschieht in den Zeilen 8 und 10 in Listing 14.3. Zeile 8 enthält die Verzweigung zum Konstruktor der Superklasse, um den Dialog mit dem Frame zu verbinden. In Zeile 10 werden die Instanzvariablen auf die jeweilige Instanz der `Frame`-Klasse gesetzt, die in der Klasse `BaseFrame2` definiert wurden.

Der Rest des `TextDialog`-Konstruktors richtet einfach nur das übrige Layout ein: ein Textfeld und eine Schaltfläche in einem `BorderLayout`. Mit der `getInsets()`-Methode werden ein paar ECKEINSÄTZE hinzugefügt, und schließlich behandelt die `action()`-Methode die Aktion der OK-Schaltfläche dieses Dialogfelds. In der `action()`-Methode passieren zwei Dinge: In Zeile 28 wird das Dialogfeld verborgen und freigegeben, und in Zeile 29 wird der Wert des Labels im Eltern-Frame auf den neuen Textwert geändert.

So viele Klassen für ein einfaches Applet! Die verschiedenen Fenster und die zugehörigen Ereignisklassen machen das Applet so kompliziert. An dieser Stelle sollten Sie aber bereits damit vertraut sein, daß jedes Teil eines Applets seine eigenen Komponenten und Aktionen hat, und wie alle diese Teile zusammenspielen. Sollte jedoch noch immer Verwirrung herrschen, können Sie den Beispielcode auf der CD durcharbeiten, um ein besseres Gefühl dafür zu bekommen, wie alles zusammengehört.

Applets und Dialogfelder zusammenfügen

Dialogfelder können nur einem `Frame` zugewiesen werden. Um ein Dialogfeld zu erstellen, müssen Sie eine Instanz der `Frame`-Klasse an eine der Konstruktor-Methoden des Dialogfelds übergeben. Das würde bedeuten, daß Sie keine Dialogfelder erstellen können, die mit Applets verbunden sind. Da Applets keine expliziten Frames haben, können Sie der `Dialog`-Klasse kein `Frame`-Argument übergeben. Aber mit ein wenig trickreichem Code können Sie eines `Frame`-Objekts habhaft werden, das das Applet enthält (häufig im Browser- oder Appletviewer-Fenster) und das Objekt dann als `Frame` für das Dialogfeld nutzen.

Bei diesem Code wird die `getParent()`-Methode eingesetzt, die für alle AWT-Komponenten definiert ist. Die `getParent()`-Methode gibt das Objekt zurück, in dem dieses Objekt enthalten ist. Das Eltern-Objekt aller AWT-Applikationen muß also ein `Frame` sein. Applets verhalten sich genauso. Durch wiederholten Aufruf von `getParent()` müßten Sie schließlich in der Lage sein, eine `Frame`-Instanz zu erhalten. Hier nun der trickreiche Code, den Sie in Ihr Applet einfügen können:

```

Object anchorpoint = getParent()
while (! (anchorpoint instanceof Frame))
    anchorpoint = ((Component)anchorpoint).getParent();

```

In der ersten Zeile erstellen Sie eine lokale Variable namens `anchorpoint`, die den schließlich gefundenen Rahmen für das Applet aufnimmt. Das Objekt, das `anchorpoint` zugewiesen wird, kann eine von vielen Klassen sein. Sie deklarieren den Typ also als `Object`.

Die zweite Zeile in diesem Code ist eine while-Schleife, die für jedes neue Objekt in der Aufwärtsskette getParent() aufruft, bis sie ein Frame-Objekt erreicht. Da die getParent() -Methode nur für Objekte definiert ist, die von Component erben, müssen Sie hier daran denken, daß Sie den Wert von anchorpoint jedesmal in Component stellen müssen, damit die getParent()-Methode funktionieren kann.

Nach dem Austritt aus der Schleife wird das Objekt in der anchorpoint-Variablen eine Instanz der Frame-Klasse (oder einer ihrer Subklassen). Anschließend können Sie ein Dialog-Objekt erstellen, das diesem Rahmen beigelegt wird, in dem Sie anchorpoint noch einmal ausgeben, um sicherzustellen, daß Sie wirklich ein Frame-Objekt haben:

```
TextDialog dl = new TextDialog((Frame)anchorpoint,
"Enter Text", true);
```

Dateidialog-Objekte

Die FileDialog-Klasse bietet ein Standard-Dialogfeld Datei öffnen/speichern, mit dem Sie auf das lokale Dateisystem zugreifen können. Diese FileDialog-Klasse ist systemunabhängig, jedoch wird je nach Plattform der normale Dialog für das Öffnen oder Speichern von Dateien angezeigt.



Bei Applets ist es vom Browser abhängig, ob Sie Instanzen von FileDialog einsetzen können. Die meisten Browser erzeugen lediglich einen Fehler, wenn Sie es versuchen. FileDialog ist bei Stand-alone-Anwendungen wesentlich nützlicher.

Um einen Dateidialog zu erstellen, benutzen Sie die folgenden Konstruktoren:

- FileDialog(Frame, String) erzeugt einen Dateidialog, der dem angegebenen Frame zugeordnet ist, mit dem angegebenen Titel. Hiermit wird ein Dialogfeld zum Öffnen einer Datei erzeugt.
- FileDialog(Frame, String, int) erzeugt ebenfalls ein Datei-Dialogfeld, jedoch wird ein Ganzzahlargument eingesetzt, um zu bestimmen, ob das Dialogfeld zum Öffnen oder Speichern einer Datei dienen soll (der einzige Unterschied besteht in der Beschriftung der Schaltflächen; der Dateidialog führt nicht das eigentliche Speichern oder Öffnen aus). Die möglichen Optionen für das Modus-Argument sind FileDialog.LOAD und FileDialog.SAVE.

Nachdem Sie eine FileDialog-Instanz erstellt haben, zeigen Sie sie mit show() an:

```
FileDialog fd = new FileDialog(this, "FileDialog");
fd.show();
```

Wenn der Benutzer eine Datei im Datei-Dialogfeld auswählt und es dann schließt, können Sie auf den vom Benutzer gewählten Dateinamen zugreifen, indem Sie die Methoden getDirectory() und getFile() verwenden. Beide Methoden geben Zeichenketten zurück, die die Werte angeben, die der Leser gewählt hat. Sie können die Datei dann mit den Stream- und File-Handler-Methoden (über die Sie nächste Woche mehr erfahren werden) öffnen und sie auslesen bzw. in sie schreiben.

Fensterereignisse

Dies sind nun die letzten Ereignisse, die Sie in AWT bearbeiten können: die Ereignisse für Fenster und Dialogfelder. (In bezug auf Ereignisse sind Dialogfeld und Fenster gleich.) Fensterereignisse entstehen, wenn sich der Zustand eines Fensters in irgendeiner Form ändert: wenn das Fenster verschoben, seine Größe geändert, auf Symbolgröße reduziert, in den Vordergrund geholt oder geschlossen wird. In einer wohlerzogenen Applikation werden Sie zumindest einige dieser Ereignisse behandeln wollen - z.B. um Threads zu stoppen, wenn ein Fenster auf Symbolgröße reduziert wird, oder für Aufräumarbeiten, wenn das Fenster geschlossen wird.

Sie können handleEvent() verwenden, um auf einzelne Ereignisse, die in Tabelle 13.8 genannt sind, zu prüfen. Dazu verwenden Sie die normale switch-Anweisung mit der Instanzvariablen id.

Tabelle 14.1: Fensterereignisse

Ereignisname	Wann es auftritt

WINDOW_DESTROY	Wird erzeugt, wenn ein Fenster über das Feld Schließen oder den Menüpunkt Schließen zerstört wird
WINDOW_EXPOSE	Wird erzeugt, wenn ein Fenster hinter einem anderen Fenster hervorgeholt wird
WINDOW_ICONIFY	Wird erzeugt, wenn das Fenster auf Symbolgröße reduziert wird
WINDOW_DEICONIFY	Wird erzeugt, wenn das Fenster aus Symbolgröße wiederhergestellt wird
WINDOW_MOVED	Wird erzeugt, wenn das Fenster verschoben wird

Menüs

Es bleibt jetzt nur noch ein Element der Benutzeroberfläche im AWT zu behandeln: Menüs.

Eine Menüleiste ist eine Zusammenstellung von Menüs. Ein Menü enthält dazu eine Zusammenstellung von Menüpunkten, die Namen haben und manchmal optional über Tastenkürzel verfügen. Das AWT bietet Klassen für alle diese Menüelemente, darunter `MenuBar`, `Menu`, und `MenuItem`.

Menüs und Menüleisten



Eine *Menüleiste* ist ein Satz von Menüs, die an der Oberkante eines Fensters erscheinen. Da sich Menüleisten immer auf Fenster beziehen, können Sie keine Menüleisten in Applets erstellen (aber wenn das Applet ein unabhängiges Fenster öffnet, kann dieses Fenster eine Menüleiste besitzen).

Um eine Menüleiste für ein bestimmtes Fenster zu erzeugen, legen Sie eine neue Instanz der Klasse `MenuBar` an:

```
MenuBar mbar = new MenuBar();
```

Um diese Menüleiste als Standardmenü des Fensters festzulegen, verwenden Sie die `setMenuBar()`-Methode (definiert in der `Frame`-Klasse):

```
window.setMenuBar(mbar);
```

Sie können individuelle Menüs (Datei, Bearbeiten etc.) in die Menüleiste einfügen, indem Sie sie erzeugen und dann in die Menüleiste mit der Methode `add()` einfügen. Das Argument für den `Menu`-Konstruktor ist der Name des Menüs, wie er in der Menüleiste auftritt.

```
Menu myMenu = new Menu("File");
mbar.add(myMenu);
```

Bei einigen Systemen kann ein spezielles Hilfemenü definiert werden, das auf der rechten Seite der Menüleiste steht. Sie können ein bestimmtes Menü mit der `setHelpMenu()`-Methode als Hilfemenü definieren. Das angegebene Menü muß bereits dem Menü selbst hinzugefügt worden sein, bevor Sie es zu einem Hilfemenü machen können.

```
Menu helpmenu = new Menu("Help");
mbar.add(helpmenu);
mbar.setHelpMenu(helpmenu);
```

Falls Sie den Benutzer aus irgendeinem Grund daran hindern möchten, ein bestimmtes Menü auszuwählen, können Sie den Befehl `disable()` auf das Menü anwenden (bzw. den `enable()`-Befehl, um es wieder verfügbar zu machen):

```
myMenu.disable();
```

Menüoptionen

Sie können vier verschiedene Optionsarten in die Menüs einfügen:

- Instanzen der Klasse `MenuItem` für normale Menüoptionen
- Instanzen der Klasse `CheckboxMenuItem` für umschaltbare Menüoptionen
- Andere Menüs mit eigenen Menüoptionen

- Trennstriche (Separatoren) zur Abgrenzung von Optionsgruppen in Menüs

Menüoptionen erstellen

Normale Menüoptionen werden in ein Menü mit der MenuItem-Klasse eingefügt. Zunächst erstellen Sie eine neue Instanz von MenuItem und fügen diese anschließend mit der add()-Methode in die Menu-Komponente ein:

```
Menu myMenu = new Menu("Tools");
myMenu.add(new MenuItem("Info"));
myMenu.add(new MenuItem("Colors"));
```

Untermenüs können einfach durch Erstellen einer neuen Instanz von Menu und deren Hinzufügen in das erste Menü erstellt werden. Danach können Sie Optionen in das Menü einfügen:

```
Menu submenu = new Menu("Sizes");
myMenu.add(submenu);
submenu.add(new MenuItem("Small"));
submenu.add(new MenuItem("Medium"));
submenu.add(new MenuItem("Large"));
```

Die CheckBoxMenuItem-Klasse erzeugt eine Menüoption mit einem Kontrollfeld, wodurch der Zustand des Menüs ein- bzw. ausgeschaltet werden kann. (Einmaliges Anklicken bewirkt die Auswahl des Kontrollfelds, nochmaliges Anklicken bewirkt die Abwahl des Kontrollfelds.) Menüoptionen mit Kontrollfeld werden auf die gleiche Art erzeugt, wie normale Menüoptionen:

```
CheckboxMenuItem coords =
new CheckboxMenuItem("Show Coordinates");
myMenu.add(coords);
```

Um schließlich einen Separator in das Menü einzufügen (eine Linie, mit der Sie Optionsgruppen in einem Menü trennen), erstellen Sie eine Menüoption mit einem einfachen Bindestrich (-) als Label und fügen diese dann ein. Die spezielle Menüoption wird als Trennlinie ausgegeben. Die beiden nächsten Zeilen Java-Code erzeugen eine Separator-Menüoption, und fügen Sie diese in das Menü myMenu ein:

```
MenuItem msep = new MenuItem("-");
myMenu.add(msep);
```

Jede Menüoption kann mit der disable()-Methode deaktiviert und mit enable() wieder aktiviert werden. Deaktivierte Menüoptionen können nicht ausgewählt werden.

```
MenuItem item = new MenuItem("Fill");
myMenu.addItem(item);
item.disable();
```

Menüereignisse

Die Auswahl einer Menüoption oder die Auswahl eines Tastenkürzels für eine Menüoption führt dazu, daß ein Aktionsereignis generiert wird. Sie können das Ereignis mit der action()-Methode behandeln, wie Sie es in den letzten Tagen gemacht haben.

Zusätzlich zu den Aktionsereignissen erzeugen CheckBoxMenuItems Listenauswahl- und Listenabwahlereignisse, die über handleEvent() behandelt werden.

Bei der Verarbeitung von Ereignissen, die von Menüoptionen und Menüoptionen mit Kontrollfeld erzeugt wurden, müssen Sie an folgendes denken: Da CheckboxMenuItem eine Subklasse von MenuItem ist, brauchen Sie diese Menüoption nicht als Sonderfall zu behandeln. Diese Aktion handhaben Sie genauso wie andere Aktionsmethoden.

AWT-Stand-alone-Applikationen erstellen

Obwohl Sie nächste Woche lernen, wie Sie grafische Benutzeroberflächen mit den neuen Swing-Klassen erstellen, besitzen Sie bereits die meisten Fähigkeiten, um eine Java-1.02-Applikation zu erstellen.

Der Grund ist, daß Sie außer in ein paar einfachen Codezeilen und in den verschiedenen Umgebungen keine großen Unterschiede zwischen einem Java-Applet und einer grafischen Java-Applikation finden werden. Alles das, was Sie bis

jetzt über das AWT gelernt haben, einschließlich der Grafikmethoden, Animationstechniken, Ereignisse, Komponenten der Benutzeroberfläche sowie Fenster und Dialogfelder, kann genauso in Java-Applikationen wie in Applets eingesetzt werden. Und Applikationen haben den Vorteil, »nicht mehr im Sandkasten zu stecken« - sie unterliegen nicht den Sicherheitsbeschränkungen wie Applets. Mit Applets können Sie (fast) alles tun, was Sie wollen.

Wie erstellen wir also eine grafische Java-Applikation? Der Code dafür ist nahezu trivial. Ihre Hauptanwendungsklasse muß von `Frame` abgeleitet sein. Werden `Threads` genutzt (für die Animation oder andere Verarbeitungsarten), muß auch `Runnable` implementiert sein:

```
class MyAWTApplication extends Frame implements Runnable {
    ...
}
```

Innerhalb der `main()`-Methode Ihrer Applikation erstellen Sie eine neue Instanz Ihrer Klasse - weil Ihre Klasse `Frame` erweitert, und damit erhalten Sie ein neues AWT-Fenster, das Sie in seiner Größe ändern und wie ein beliebiges anderes AWT-Fenster anzeigen können.

In der Konstruktor-Methode Ihrer Klasse richten Sie die üblichen AWT-Funktionen für Fenster ein, wie Sie es normalerweise in der `init()`-Methode eines Applets täten: Legen Sie den Titel fest, fügen Sie den `Layout-Manager` ein, erstellen Sie Komponenten wie eine Menüleiste oder andere Elemente der Benutzeroberfläche und fügen sie ein, starten Sie ein `Thread` etc.

Hier ein ganz einfaches Applikationsbeispiel:

```
import java.awt.*;

class MyAWTApplication extends Frame {

    MyAWTApplication(String title) {
        super(title);
        setLayout(new FlowLayout());
        add(new Button("OK"));
        add(new Button("Reset"));
        add(new Button("Cancel"));
    }

    public static void main(String args[]) {
        MyAWTApplication app = new MyAWTApplication("Hi! I'm an application");
        app.resize(300,300);
        app.show();
    }
}
```

Meistens können Sie die diese Woche erlernten Methoden einsetzen, um Ihre Applikation zu steuern und zu verwalten. Nur die für Applets spezifischen Methoden können Sie nicht verwenden (d.h. solche, die in `java.applet.Applet` definiert sind, darunter Methoden zum Erhalt von `URL`-Informationen und zum Abspielen von `Audio-Clips`). Weitere Einzelheiten hierzu finden Sie in der `API-Dokumentation` zu dieser Klasse.

Und noch einen Unterschied zwischen Applikationen und Applets sollten Sie kennen: Wenn Sie ein `Fensterschließereignis` behandeln, müssen Sie neben dem Verbergen oder Zerstören des Fensters auch `System.exit(0)` aufrufen, um das System zu informieren, daß Ihre Applikation beendet wurde:

```
public void windowClosing(WindowEvent e) {
    win.hide();
    win.destroy();
    System.exit(0);
}
```

Komplettes Beispiel: RGB/HSB-Konverter

Wir haben bisher viel Theorie und kleine Beispiele durchgearbeitet und wenden uns nun einem größeren Beispiel zu, in dem die bisher gelernten Teile zusammengesetzt werden. Das folgende Beispiel-Applet zeigt die Erstellung des Layouts,

das Verschachteln von Panels, Erstellen von Komponenten der Benutzeroberfläche und die Ereignisbehandlung sowie den Einsatz mehrerer Klassen, die in einem einzelnen Applet zusammengesetzt werden. Kurz gesagt: das ist das komplexeste Applet, das Sie bisher erstellt haben.

Abbildung 14.3 zeigt das Applet, das Sie in diesem Beispiel erstellen. Das ColorTest- Applet ermöglicht es Ihnen, Farben auf der Basis von RGB-Werten (Rot, Grün und Blau) und HSB-Werten (Farbton, Sättigung und Helligkeit) auszuwählen.

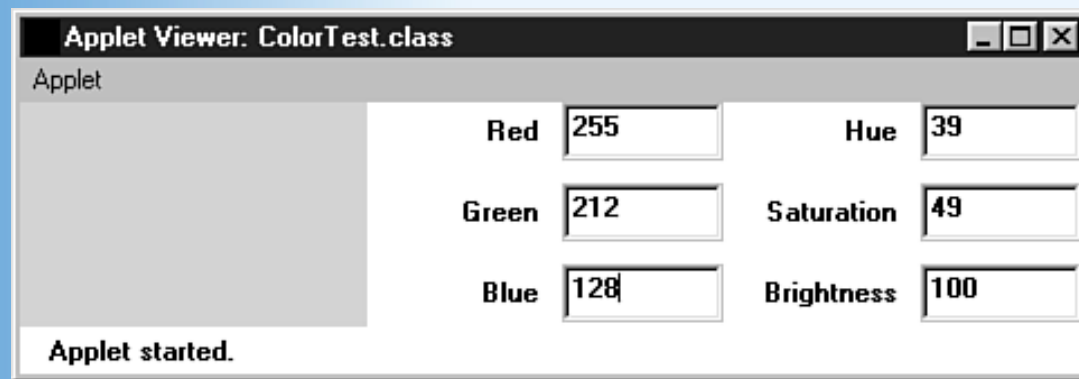


Abbildung 14.3:
Das ColorTest-Applet



Schnell eine kurze Zusammenfassung zur Farbtheorie, für den Fall, daß Sie nicht damit vertraut sind: RGB definiert eine Farbe nach ihren Rot-, Grün- und Blau-Werten. Kombinationen dieser Werte können fast jede beliebige Farbe des Spektrums erzeugen. (Rot, Grün und Blau sind sogenannte additive Farben; so werden z.B. verschiedene Farben auf Ihrem Bildschirm und Fernseher dargestellt.)

HSB steht für Farbton (Hue), Sättigung (Saturation) und Helligkeit (Brightness) und bildet eine andere Art der Farbangabe. Der Farbton ist die eigentliche Farbe im Spektrum (stellen Sie sich das als Wert auf einem Farbrad vor). Die Sättigung ist die Menge dieser Farbe: eine geringe Sättigung ergibt Pastellfarben, Farben mit hoher Sättigung sind lebendiger und »farbiger«. Helligkeit ist die Helligkeit bzw. Dunkelheit der Farbe. Keine Helligkeit ist Schwarz, volle Helligkeit ist Weiß.

Eine einzelne Farbe kann entweder durch ihre RGB-Werte oder durch ihre HSB-Werte dargestellt werden, und mathematische Algorithmen können eine Konvertierung zwischen beiden Werten vornehmen. Das ColorTest-Applet liefert einen grafischen Konverter zwischen diesen beiden.

Das ColorTest-Applet hat drei Hauptteile: eine Farbbox auf der linken Seite und zwei Gruppen mit Textfeldern auf der rechten Seite. Die erste Feldergruppe zeigt die RGB- Werte, die rechte die HSB-Werte an. Durch Ändern der Werte in einem der Textfelder wird die Farbbox aktualisiert, so daß jeweils die Farbe angezeigt wird, die in den Feldern gewählt wurde.

Dieses Applet nutzt zwei Klassen:

- ColorTest: Diese Klasse ist von Applet abgeleitet. Sie steuert das Applet selbst.
- ColorControls: Diese Klasse ist von Panel abgeleitet. Sie erzeugen diese Klasse, um eine Gruppe von drei Textfeldern darzustellen und die Aktionen dieser Textfelder zu handhaben. Ferner werden zwei Instanzen dieser Klasse, eine für die RGB- Werte und eine für die HSB-Werte, erstellt und in das Applet eingefügt.

Arbeiten wir nun das Beispiel Schritt für Schritt durch, weil es kompliziert ist und leicht verwirren kann. Am Ende dieser Lektion steht der gesamte Code dieses Applets.

Entwerfen und Erstellen des Applet-Layouts

Die beste Art, mit einem Applet zu beginnen, das AWT-Komponenten beinhaltet, ist, sich zuerst um das Layout und dann um die Funktionalität zu kümmern. Beim Layout sollten Sie mit dem äußersten Panel beginnen und sich dann nach innen durcharbeiten.

Sie können sich die Arbeit vereinfachen, indem Sie alle Panels Ihres Benutzeroberflächendesigns zuerst auf Papier aufzeichnen. Das hilft Ihnen bei der Anordnung der Panels innerhalb des Applets oder Fensters und der besten Ausnutzung von Layout und Platz. Papierentwürfe sind selbst dann hilfreich, wenn Sie kein GridBagLayout verwenden, aber sicherlich nützlich, wenn Sie ein GridBagLayout nutzen. (Für dieses Applet nutzen Sie ein einfaches GridLayout.)

Abbildung 14.4 zeigt das ColorTest-Applet mit einem darübergelegten Raster, so daß Sie eine Vorstellung davon bekommen können, wie die Panels und die eingebetteten Panels funktionieren.

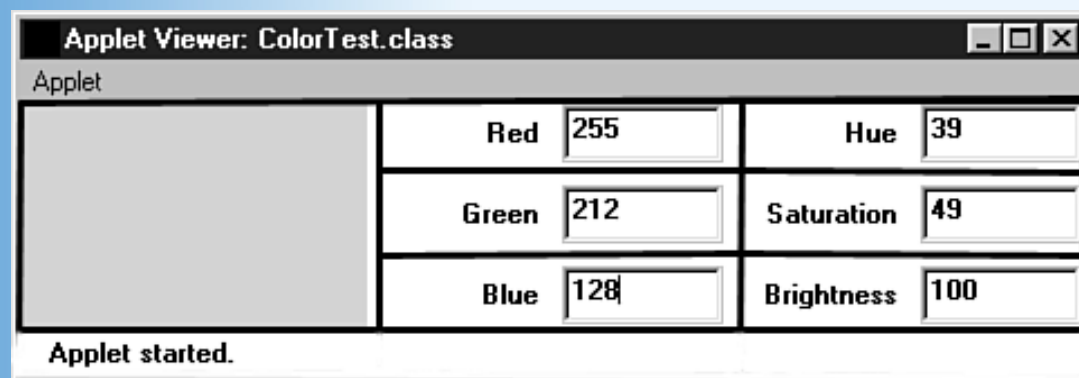


Abbildung 14.4:
Die Panels und Komponenten des ColorTest-Applets

Beginnen wir mit dem äußersten Panel - dem Applet selbst. Dieses Panel besteht aus drei Teilen: der Farbbox auf der linken Seite, den RGB-Textfeldern in der Mitte und den HSB-Feldern auf der rechten Seite.

Da das äußerste Panel das Applet selbst ist, ist die ColorTest-Klasse die Klasse des Applets. Sie wird von Applet abgeleitet. Außerdem importieren Sie hier die AWT- Klassen. (Beachten Sie, daß wir der Einfachheit halber das komplette Paket importieren, da wir hier soviel davon nutzen.)

```
import java.awt.*;
```

```
public class ColorTest extends java.applet.Applet {
    //...
}
```

Dies Applet hat drei Hauptelemente, die verfolgt werden müssen: die Farbbox und die beiden untergeordneten Panels. Jedes dieser beiden Sub-Panels bezieht sich auf etwas anderes, aber grundsätzlich sind sie gleich und verhalten sich gleich. Anstatt hier viel Code in dieser Klasse zu kopieren, können Sie die Gelegenheit nutzen und eine weitere Klasse für diese Sub-Panels erstellen, wobei Sie Instanzen dieser Klasse hier im Applet verwenden und alles miteinander über Methoden kommunizieren lassen. Schon bald definieren wir die neue Klasse namens ColorControls.

Jetzt müssen Sie jedoch erst wissen, wie Sie alle drei Teile des Applets handhaben, damit Sie sie bei Änderungen aktualisieren können. Erstellen wir also drei Instanzvariablen: eine vom Typ Canvas für die Farbbox und die anderen beiden vom Typ ColorControls für die Kontroll-Panels:

```
ColorControls RGBcontrols, HSBcontrols;
Canvas swatch;
```

Jetzt können Sie mit der init()-Methode fortfahren, in der die gesamte Initialisierung und das Layout stattfinden. Sie arbeiten in drei Schritten:

1. Erstellen Sie das Layout für die großen Teile des Panels. Ein FlowLayout wäre zwar möglich, jedoch eignet sich ein GridLayout mit einer Zeile und drei Spalten besser.
2. Erstellen und initialisieren Sie die drei Komponenten dieses Applets: ein Zeichenbereich für die Farbbox und zwei Sub-Panels für die Textfelder.
3. Fügen Sie diese Komponenten in das Applet ein.

Der erste Schritt ist das Layout. Benutzen Sie ein GridLayout mit einem Abstand von 10 Punkten, um die Komponenten voneinander zu trennen:

```
setLayout(new GridLayout(1, 3, 5, 10));
```

Der zweite Schritt ist das Erstellen der Komponenten - zuerst den Zeichenbereich. Sie haben eine Instanzvariable, die diesen enthält. Jetzt erstellen Sie den Zeichenbereich und initialisieren den Hintergrund mit Schwarz:

```
swatch = new Canvas();
swatch.setBackground(Color.black);
```

Sie müssen hier auch zwei Instanzen des noch nicht existierenden ColorControls-Panels einfügen. Da Sie diese Klasse noch nicht angelegt haben, wissen Sie nicht, wie die Konstruktoren für diese Klasse aussehen werden. In diesem Fall fügen Sie hier einfach ein paar Platzhalter-Konstruktoren ein und füllen die Einzelheiten später aus.

```
RGBcontrols = new ColorControls(...)
HSBcontrols = new ColorControls(...);
```

Der dritte Schritt ist das Einfügen aller drei Komponenten in das Applet-Panel, und zwar so:

```
add(swatch);
add(RGBcontrols);
add(HSBcontrols);
```

Da Sie gerade am Layout arbeiten, fügen Sie als zusätzlichen Freiraum einen Eckeinsatz ein - 10 Punkt an allen Kanten:

```
public Insets getInsets() {
return new Insets(10, 10, 10, 10);
}
```

Soweit mitgekommen? Dann müßten Sie jetzt drei Instanzvariablen, eine init()- Methode mit zwei unvollständigen Konstruktoren und eine getInsets()-Methode in Ihrer ColorTest-Klasse haben. Fahren wir nun fort mit der Erstellung des Layouts für die Subpanels in der ColorControls-Klasse, damit Sie die Konstruktoren angeben und das Layout abschließen können.

Definieren der untergeordneten Panels

Die ColorControls-Klasse enthält das Verhalten für das Layout und die Handhabung der Sub-Panels, die die RGB- und HSB-Farbwerte darstellen. ColorControls braucht keine Subklasse von Applet zu sein, da es sich nicht um ein Applet, sondern ein Panel handelt. Definieren Sie es so, daß es eine Subklasse von Panel ist:

```
import java.awt.*

class ColorControls extends Panel {
...
}
```

Die ColorControls-Klasse benötigt eine Reihe von Instanzvariablen, damit die Informationen vom Panel zurück zum Applet gelangen können. Die erste dieser Instanzvariablen ist eine Rückverzweigung zur Klasse des Applets, die dieses Panel enthält. Da die äußere Applet-Klasse die Aktualisierung der einzelnen Panels steuert, muß dieses Panel eine Möglichkeit haben, dem Applet mitzuteilen, daß sich etwas geändert hat. Und um eine Methode in dem Applet aufzurufen, müssen Sie einen Bezug zum Objekt herstellen. Also ist die erste Instanzvariable eine Referenz zu einer Instanz der Klasse ColorTest:

```
ColorTest applet;
```

Stellen Sie sich vor, daß die Applet-Klasse alles aktualisiert, dann ist diese Klasse an den einzelnen Textfeldern in diesem Sub-Panel interessiert. Sie müssen also für jedes dieser Textfelder eine Instanzvariable erstellen:

```
TextField tfield1, tfield2, tfield3;
```

Jetzt können Sie sich an den Konstruktor für diese Klasse machen. Da diese Klasse kein Applet ist, verwenden Sie zur Initialisierung nicht init(), sondern eine Konstruktor-Methode. Innerhalb des Konstruktors machen Sie das, was Sie in init() machen würden: das Layout für das untergeordnete Panel sowie die Textfelder erstellen und sie in das Panel einfügen.

Das Ziel ist hier, die ColorControls-Klasse allgemein genug zu halten, so daß Sie sie sowohl für die RGB- als auch für die HSB-Felder nutzen können. Diese beiden Panels unterscheiden sich nur durch die Beschriftung für den Text. Das sind die drei Werte, die Sie haben müssen, bevor Sie das Objekt erzeugen können. Sie können diese drei Werte über die Konstruktoren in ColorTest übergeben. Aber Sie brauchen noch mehr: den Bezug zum übergeordneten Applet, den Sie ebenfalls über den Konstruktor erhalten können.

Damit haben Sie jetzt vier Argumente für den grundlegenden Konstruktor der `ColorControls` -Klasse. Die Signatur für den Konstruktor sieht so aus:

```
ColorControls(ColorTest parent,
String l1, String l2, String l3) {
}
```

Lassen Sie uns mit der Definition dieses Konstruktors beginnen, indem Sie zunächst der `applet`-Instanzvariablen den Wert von `parent` zuweisen:

```
applet = parent;
```

Als nächstes erstellen Sie das Layout für dieses Panel. Sie können hier auch wieder, wie beim `Applet`-Panel, ein `GridLayout` für diese Sub-Panels verwenden, aber diesmal hat das Raster drei Zeilen (je eine für jedes Textfeld mit Beschriftung) und zwei Spalten (eine für die Beschriftung und eine für die Felder). Und definieren Sie wieder einen Abstand von 10 Pixeln zwischen den Komponenten im Raster:

```
setLayout(new GridLayout(3,2,10,10));
```

Jetzt können Sie die Komponenten erstellen und in das Panel einfügen. Zuerst erstellen Sie die Textfeld-Objekte (initialisiert mit dem String "0") und weisen sie den entsprechenden Instanzvariablen zu:

```
tfield1 = new TextField("0");
tfield2 = new TextField("0");
tfield3 = new TextField("0");
```

Dann fügen Sie diese Felder und die zugehörigen Label in das Panel ein, wobei Sie die verbleibenden Parameter im Konstruktor als Beschriftungstext in den Labels verwenden:

```
add(new Label(l1, Label.RIGHT));
add(tfield1);
add(new Label(l2, Label.RIGHT));
add(tfield2);
add(new Label(l3, Label.RIGHT));
add(tfield3);
```

Damit ist der Konstruktor für die Sub-Panel-Klasse `ColorControls` abgeschlossen. Sind Sie mit dem Layout fertig? Noch nicht ganz. Fügen Sie noch einen Eckeinsatz im Sub-Panel ein - nur an der oberen und unteren Kante - das sieht besser aus. Die Eckeinsätze fügen Sie genauso ein, wie Sie es schon bei der `ColorTest`-Klasse getan haben, nämlich mit der `getInsets()`-Methode:

```
public Insets getInsets() {
return new Insets(10, 10, 0, 0);
}
```

Sie haben es fast geschafft. 98% der Grundstruktur sind fertig, und es ist nur noch ein Schritt auszuführen. Dazu gehen Sie zurück zu `ColorTest` und ergänzen die Platzhalter-Konstruktoren für das Sub-Panel, damit sie mit dem tatsächlichen Konstruktor für `ColorControls` übereinstimmen.

Der Konstruktor für `ColorControls`, den Sie gerade erstellt haben, hat vier Argumente: das `ColorTest`-Objekt und drei Labels (Zeichenketten). Erinnern Sie sich noch, wie Sie die `init()`-Methode für `ColorTest` erstellt haben? Sie haben zwei Platzhalter für das Erstellen neuer `ColorControls`-Objekte eingefügt. Diese Platzhalter ersetzen Sie jetzt mit der richtigen Version. Und vergessen Sie dabei nicht, die vier Argumente einzufügen, die der Konstruktor braucht: das `ColorTest`-Objekt und die drei Strings. Das `ColorTest`-Objekt können Sie an diese Konstruktor-Aufrufe mit dem Schlüsselwort `this` übergeben:

```
RGBcontrols = new ColorControls(this, "Red", "Green", "Blue");
HSBcontrols = new ColorControls(this, "Hue", "Saturation", "Brightness");
```



Bei allen Initialisierungswerten in diesem Beispiel habe ich die Zahl 0 gewählt (eigentlich den String "0"). Für die Farbe Schwarz ist sowohl der RGB- als auch der HSB- Wert 0, und daher habe ich diese Vorgabe gewählt. Wenn Sie das Applet

mit einer anderen Farbe initialisieren wollen, können Sie die ColorControls-Klasse so umschreiben, daß sie auch Initialisierungswerte nutzt, um die Label zu initialisieren. Dies hier war nur ein kürzeres Beispiel.

Ereignisbehandlung

Mit dem fertigen Layout können Sie die Ereignisbehandlung und die Aktualisierungsvorgänge zwischen den verschiedenen Komponenten festlegen, so daß das Applet reagieren kann, wenn der Benutzer mit dem Applet interagiert.

Die Aktionsereignisse des Applets treten auf, wenn der Benutzer in einem der Textfelder einen Wert ändert. Durch das Aktionsereignis eines Textfelds wird die Farbe verändert, das Farbfeld mit der neuen Farbe entsprechend aktualisiert und in den Feldern des anderen Sub-Panels werden die Werte angezeigt, die die neue Farbe repräsentieren.

Die übergeordnete ColorTest-Klasse ist eigentlich für die Aktualisierung zuständig, da Sie alle Sub-Panels überwacht. Sie sollten allerdings Ereignisse in dem Subpanel verfolgen und annehmen, in denen diese auftreten. Da sich das eigentliche Aktionsereignis des Applets auf ein Textfeld bezieht, können Sie die Methode action() verwenden, um das Ereignis in der ColorControls-Klasse abzufangen:

```
public boolean action(Event evt, Object arg) {
    if (evt.target instanceof TextField) {
        applet.update(this);
        return true;
    }
    else return false;
}
```

In der action()-Methode stellen Sie sicher, daß das Aktionsereignis tatsächlich von einem Textfeld ausgelöst wurde (da lediglich Textfelder zur Verfügung stehen, können Aktionsereignisse auch nur von diesen stammen; es ist aber generell sinnvoll, einen derartigen Test durchzuführen). Wenn dem so ist, rufen Sie die Methode update() auf, die in der Klasse ColorTest definiert ist, um das Applet mit den neuen Werten zu aktualisieren. Da das äußere Applet für die gesamte Aktualisierung zuständig ist, benötigen Sie diese Verbindung zurück zum Applet - auf diese Weise können Sie die richtige Methode zur richtigen Zeit aufrufen.

Aktualisieren des Ergebnisses

Jetzt kommt der schwierige Teil: die eigentliche Aktualisierung auf der Basis der neuen Werte, gleichgültig welches Textfeld geändert wurde. Bei diesem Schritt definieren Sie die update()-Methode in der ColorTest-Klasse. Diese update()-Methode nimmt ein einzelnes Argument an: die ColorControls-Instanz, die den geänderten Wert enthält. (Sie erhalten das Argument aus den Ereignismethoden im ColorControls-Objekt.)



Fürchten Sie nicht, daß diese update()-Methode die update()-Methode des Systems stört? Sie tut es nicht, denn wie Sie sich erinnern, können Methoden den gleichen Namen, jedoch unterschiedliche Signaturen und Definitionen haben. Da diese update()-Methode nur ein Argument vom Typ ColorControls hat, beeinflußt sie nicht die übrigen update()-Versionen. Normalerweise sollten alle Methoden namens update() im Grunde dieselbe Aufgabe haben. Das ist hier nicht der Fall, aber das ist nur ein Beispiel.

Die update()-Methode ist für das Aktualisieren aller Panels im Applet zuständig. Um zu wissen, welches Panel zu aktualisieren ist, müssen Sie wissen, welches Panel geändert wurde. Das finden Sie heraus, indem Sie testen, ob das übergebene Argument mit den untergeordneten Panels, die Sie in den Instanzvariablen RGBcontrols und HSBcontrols gespeichert haben, identisch ist:

```
void update(ColorControls controlPanel) {

if (controlPanel == RGBcontrols) { // RGB geändert, HSB aktualisieren
...
} else { // HSB geändert, RGB aktualisieren
...
}
}
```


Dieser Test ist der Kern der `update()`-Methode. Beginnen wir mit dem ersten Fall - einer Zahl, die in den RGB-Textfeldern geändert wurde. Anhand dieser neuen RGB- Werte müssen Sie ein neues `Color`-Objekt erstellen und die Werte im HSB-Panel aktualisieren. Damit Sie nicht so viel tippen müssen, können Sie ein paar lokale Variablen deklarieren, in die Sie einige Grundwerte stellen. Die Werte der Textfelder sind Zeichenketten, deren Werte Sie dazu heranziehen können, die `getText()`-Methode zu nutzen, die in den `TextField`-Objekten des `ColorControls`-Objekts definiert wurden. Da Sie bei dieser Methode meistens mit Ganzzahlwerten arbeiten, können Sie diese Zeichenketten abfragen, sie in Ganzzahlen konvertieren und in lokalen Variablen speichern (`value1`, `value2`, `value3`). Mit dem folgenden Code erledigen Sie diese Aufgabe (das sieht komplizierter aus als es ist):

```
int value1 = Integer.parseInt(controlPanel.textField1.getText());
int value2 = Integer.parseInt(controlPanel.textField2.getText());
int value3 = Integer.parseInt(controlPanel.textField3.getText());
```

Wenn Sie die lokalen Variablen definieren, brauchen Sie auch eine für das neue `Color` -Objekt:

```
Color c;
```

Nehmen wir an, eines der Textfelder auf der RGB-Seite des Applets hat sich geändert, und Sie fügen den Code in den `if`-Teil der `update()`-Methode ein. Sie müssen ein neues `Color`-Objekt erstellen und die HSB-Seite des Panels aktualisieren. Dieser erste Teil ist einfach. Bei drei RGB-Werten können Sie ein neues `Color`-Objekt erstellen, wobei Sie diese Werte als Argumente für den Konstruktor nutzen:

```
c = new Color(value1, value2, value3);
```



Dieser Teil des Beispiels ist nicht sehr robust. Er basiert auf der Annahme, daß der Benutzer nichts anderes als ganze Zahlen zwischen 0 und 255 in die Textfelder eingibt. Eine bessere Version wäre ein Test, mit dem sichergestellt wird, daß keine Dateneingabefehler passieren. Aber ich wollte das Beispiel kurz halten.

Jetzt konvertieren Sie die RGB-Werte nach HSB. Standardalgorithmen können eine auf RGB basierende Farbe in eine HSB-Farbe konvertieren, aber Sie brauchen sie nicht nachzusehen. Die `Color`-Klasse verfügt über eine Klassenmethode namens `RGBtoHSB()`, die Sie benutzen können. Diese Methode erledigt die Arbeit für Sie - oder zumindest die meiste Arbeit. Die `RGBtoHSB()`-Methode wirft jedoch zwei Probleme auf:

- Die `RGBtoHSB()`-Methode gibt ein Array mit den HSB-Werten zurück, so daß Sie diese Werte aus dem Array extrahieren müssen.
- Die HSB-Werte sind in Fließkommawerten zwischen 0.0 bis 1.0 angegeben. Ich persönlich stelle mir HSB-Werte lieber als Ganzzahlen vor, wobei ein Farbton eine Gradzahl auf einem Farbrad (0 bis 360 Grad) und Sättigung sowie Helligkeit Prozentwerte zwischen 0 und 100 sind.

Aber keines dieser Probleme ist unüberwindbar. Sie müssen lediglich einige Zeilen Code hinzufügen. Beginnen wir mit dem Aufruf von `RGBtoHSB()` mit den neuen RGB- Werten. Die Methode gibt ein Array mit floats zurück, und daher müssen Sie eine lokale Variable (`HSB`) erstellen, die die Ergebnisse der `RGBtoHSB()`-Methode speichert. (Denken Sie daran, daß Sie auch ein leeres float-Array als viertes Argument für `RGBtoHSB()` erstellen und übergeben müssen.)

```
float[] HSB = Color.RGBtoHSB(value1, value2, value3, (new float[3]));
```

Jetzt konvertieren Sie diese Fließkommawerte, die zwischen 0.0 und 1.0 liegen, in Werte zwischen 0 und 100 (für Sättigung und Helligkeit) bzw. 0 und 360 für den Farbton, indem Sie die entsprechenden Zahlen multiplizieren und die Werte dem array wieder zuweisen:

```
HSB[0] *= 360;
HSB[1] *= 100;
HSB[2] *= 100;
```

Jetzt haben Sie alle gewünschten Zahlen. Der letzte Teil der Aktualisierung ist, diese Werte wieder in die Textfelder einzutragen. Natürlich sind diese Werte noch immer Fließkommazahlen, und Sie müssen sie in `int`-Werte casten, bevor sie in Zeichenketten umgewandelt und gespeichert werden:

```
HSBcontrols.textField1.setText(String.valueOf((int)HSB[0]));
HSBcontrols.textField2.setText(String.valueOf((int)HSB[1]));
HSBcontrols.textField3.setText(String.valueOf((int)HSB[2]));
```

Die Hälfte haben Sie geschafft. Der nächste Teil des Applets ist derjenige, der die RGB-Werte aktualisiert, wenn sich ein Textfeld auf der HSB-Seite geändert hat. Das geschieht im else-Teil des großen if...else-Abschnitts, wo diese Methode definiert und festgelegt wird, was aktualisiert wird.

Es ist eigentlich einfacher, RGB-Werte aus HSB-Werten zu erzeugen, als andersherum. Eine Klassenmethode in der Color-Klasse, die getHSBColor()-Methode, erzeugt ein neues Color-Objekt aus den drei HSB-Werten. Wenn Sie ein Color-Objekt haben, können Sie die RGB-Werte daraus leicht herausziehen. Der Trick ist natürlich, daß getHSBColor drei Fließkommargumente annimmt, wohingegen die Werte, die Sie haben, Ganzzahlwerte sind, mit denen ich lieber arbeite. Bei diesem getHSBColor()- Aufruf müssen Sie jetzt die Ganzzahlwerte aus den Textfeldern in floats-Werte casten und sie durch den entsprechenden Konvertierungsfaktor dividieren. Das Ergebnis von getHSBColor ist ein Color-Objekt, und deshalb können Sie das Objekt einfach der lokalen Variablen c zuweisen, damit Sie es später weiterverwenden können:

```
c = Color.getHSBColor((float)value1 / 360,
(float)value2 / 100, (float)value3 / 100);
```

Sind alle Elemente des Color-Objekts gesetzt, müssen die RGB-Werte für das Aktualisieren aus dem Color-Objekt extrahiert werden. Diese Arbeit erledigen für Sie die Methoden getRed(), getGreen() und getBlue(), die in der Color-Klasse definiert sind:

```
RGBcontrols.tfield1.setText(String.valueOf(c.getRed()));
RGBcontrols.tfield2.setText(String.valueOf(c.getGreen()));
RGBcontrols.tfield3.setText(String.valueOf(c.getBlue()));
```

Schließlich müssen Sie noch, unabhängig davon, ob sich der RGB- oder HSB-Wert geändert hat, die Farbbox auf der linken Seite aktualisieren, damit sie die neue Farbe darstellt. Da das neue Color-Objekt in der Variablen c gespeichert ist, können Sie die setBackground-Methode zur Änderung der Farbe nutzen. Denken Sie aber daran, daß setBackground den Bildschirm nicht automatisch neu zeichnet, so daß Sie auch ein repaint() aufrufen müssen:

```
swatch.setBackground(c);
swatch.repaint();
```

Das war's! Jetzt kompilieren Sie die beiden Klassen ColorTest und ColorControls, erstellen eine HTML-Datei, um das ColorTest-Applet zu laden, und probieren es aus.

Der komplette Quellcode

Listing 14.4 enthält den vollständigen Quellcode für die Applet-Klasse ColorTest, und Listing 14.5 zeigt den Quellcode für die Hilfsklasse ColorControls. Meist ist es anhand eines kompletten Quelltextes einfacher, sich vorzustellen, was in einem Applet abläuft - wenn alles zusammensteht und Sie die Methodenaufrufe nachvollziehen und sehen können, welche Werte übergeben werden. Beginnen wir mit der init()-Methode im ColorTest-Applet.

Listing 14.4: Der gesamte Quelltext von ColorTest.java

```
1: import java.awt.*;
2:
3: public class ColorTest extends java.applet.Applet {
4:     ColorControls RGBcontrols, HSBcontrols;
5:     Canvas swatch;
6:
7:     public void init() {
8:         setLayout(new GridLayout(1, 3, 5, 15));
9:         swatch = new Canvas();
10:        swatch.setBackground(Color.black);
11:        RGBcontrols = new ColorControls(this, "Red",
12:            "Green", "Blue");
13:        HSBcontrols = new ColorControls(this, "Hue",
14:            "Saturation", "Brightness");
15:        add(swatch);
16:        add(RGBcontrols);
17:        add(HSBcontrols);
18:    }
19:
20:    public Insets getInsets() {
```

```

21:         return new Insets(10, 10, 10, 10);
22:     }
23:
24:     void update(ColorControls controlPanel) {
25:         int value1 = Integer.parseInt(controlPanel.tfield1.getText());
26:         int value2 = Integer.parseInt(controlPanel.tfield2.getText());
27:         int value3 = Integer.parseInt(controlPanel.tfield3.getText());
28:         Color c;
29:         if (controlPanel == RGBcontrols) { // RGB geändert, HSB aktualisieren
30:             c = new Color(value1, value2, value3);
31:             float[] HSB = Color.RGBtoHSB(value1, value2,
32:                 value3, (new float[3]));
33:             HSB[0] *= 360;
34:             HSB[1] *= 100;
35:             HSB[2] *= 100;
36:             HSBcontrols.tfield1.setText(String.valueOf((int)HSB[0]));
37:             HSBcontrols.tfield2.setText(String.valueOf((int)HSB[1]));
38:             HSBcontrols.tfield3.setText(String.valueOf((int)HSB[2]));
39:         } else { // HSB geändert, RGB aktualisieren
40:             c = Color.getHSBColor((float)value1 / 360,
41:                 (float)value2 / 100, (float)value3 / 100);
42:             RGBcontrols.tfield1.setText(String.valueOf(c.getRed()));
43:             RGBcontrols.tfield2.setText(String.valueOf(c.getGreen()));
44:             RGBcontrols.tfield3.setText(String.valueOf(c.getBlue()));
45:         }
46:         swatch.setBackground(c);
47:         swatch.repaint();
48:     }
49: }

```

Listing 14.5: Der gesamte Quelltext von ColorControls.java

```

1: import java.awt.*;
2:
3: class ColorControls extends Panel {
4:     ColorTest applet;
5:     TextField tfield1, tfield2, tfield3;
6:
7:     ColorControls(ColorTest parent,
8:         String l1, String l2, String l3) {
9:
10:         applet = parent;
11:         setLayout(new GridLayout(3,2,10,10));
12:         tfield1 = new TextField("0");
13:         tfield2 = new TextField("0");
14:         tfield3 = new TextField("0");
15:         add(new Label(l1, Label.RIGHT));
16:         add(tfield1);
17:         add(new Label(l2, Label.RIGHT));
18:         add(tfield2);
19:         add(new Label(l3, Label.RIGHT));
20:         add(tfield3);
21:
22:     }
23:
24:     public Insets getInsets() {
25:         return new Insets(10, 10, 0, 0);
26:     }
27:

```

```

28:      public boolean action(Event evt, Object arg) {
29:          if (evt.target instanceof TextField) {
30:              applet.update(this);
31:              return true;
32:          }
33:          else return false;
34:      }
35:  }

```

Nachdem Sie diese beiden Klassen kompiliert haben, kann das ColorTest-Applet auf einer Webseite mit dem folgenden HTML-Code geladen werden:

```

<applet code="ColorTest.class" width=475 height=100>
</applet>

```

Zusammenfassung

Vier Tage lang sich auf ganz bestimmte Elemente der Sprache Java zu konzentrieren ist eine lange Zeit. Das Abstract Windowing Toolkit ist allerdings ein wesentlicher Bestandteil des Werkzeugkastens eines Java-Programmierers.

Sie sind nun in der Lage, grafische Benutzeroberflächen für ein Applet zu entwerfen oder können sogar eine Applikation erstellen, die die Techniken des AWT und von Java 1.02 verwendet. In der letzten Woche des Buches werden Sie lernen, wie Sie einige dieser Aufgaben mit den Swing-Klassen von Java 1.2 bewältigen.

Egal, ob mit Tränen in den Augen oder freudestrahlend Abschied nehmen, werden Sie morgen vom AWT ablassen und beginnend mit dem morgigen Tag sich neuen Themen zuwenden.

Gute Arbeit! - Dieses Lob haben Sie sich redlich verdient.

Fragen und Antworten

Frage:

Bei der Diskussion der Stand-alone-Applikationen habe ich den Eindruck gewonnen, daß absolut kein Unterschied zwischen einem Applet und einer Applikation besteht. Wieso?

Antwort:

Sowohl Applets als auch Applikationen verwenden genau dieselben Prozeduren im AWT, um Komponenten aufzubauen und anzuzeigen sowie Ereignisse zu behandeln. Die Unterschiede liegen nur darin, daß Applikationen von main() initialisiert und in ihrem eigenen Fenster angezeigt werden, und daß Applets von init() und start() initialisiert bzw. gestartet werden. Bei der großen Ähnlichkeit zwischen Applets und Applikationen, können Sie 99% dessen, was Sie hinsichtlich Applets gelernt haben, bei Applikationen verwenden. Und weil Applets die main()-Methode ignorieren, sollte sie in einer Klasse existieren, gibt es keinen Grund, daß Sie nicht ein einzelnes Programm erstellen, das genausogut als Applet und als Applikation läuft.

Frage:

Ich habe eine Stand-alone-Applikation erstellt. Aber wenn ich auf das Schließen- Schaltfeld klicke, passiert nichts. Was muß ich machen, damit meine Anwendung wirklich beendet wird?

Antwort:

Behandeln Sie im 1.02-Ereignismodell das Ereignis WINDOW_CLOSE. Als Reaktion auf dieses Ereignis rufen Sie hide() auf, wenn das Fenster später wieder auftreten kann, oder rufen Sie destroy() auf, um es ein für alle Mal loszuwerden. Führt das Fensterschließereignis zum kompletten Beenden Ihrer Applikation, rufen Sie auch System.exit() auf.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 3

Tag 15

Pakete, Schnittstellen und mehr

Die dritte Woche dieses Kurses erweitert, was Sie bereits wissen. Sie könnten an dieser Stelle aufhören und sinnvolle Programme entwickeln. Allerdings würden Ihnen einige der fortgeschritteneren Features, die die Stärke der Sprache ausmachen, fehlen.

Heute werden Sie Ihr Wissen über Klassen, und wie diese mit anderen Klassen in einem Programm interagieren, ausbauen. Die folgenden Themen werden behandelt:

- Kontrollieren des Zugriffs auf Methoden und Variablen einer Klasse von außen
- Klassen, Methoden und Variablen mit dem Schlüsselwort `finalize` - so schützen Sie Klassen vor dem Ableiten, Methoden vor dem Überschreiben und den Wert einer Variablen davor, verändert zu werden
- Erstellung von abstrakten Klassen und Methoden, um gemeinsame Verhaltensweisen in Superklassen zusammenzufassen
- Klassen in Paketen gruppieren
- Mit Schnittstellen Lücken in der Klassenhierarchie schließen

Modifier

Die Techniken, die Sie heute für die Programmierung lernen, schließen unterschiedliche Strategien und Denkansätze zur Organisation von Klassen ein. Eine Sache haben allerdings alle diese Techniken gemein: sie verwenden alle spezielle Schlüsselworte von Java - die Modifier.

In der ersten Woche haben Sie gelernt, wie Sie in Java Klassen, Methoden und Variablen definieren. Modifier sind Schlüsselworte, die Sie den Definitionen hinzufügen, um deren Bedeutung zu verändern.

Java bietet eine große Auswahl an Modifiern an, darunter:

- Modifier für die Kontrolle des Zugriffs auf eine Klasse, Methode oder Variable: `public`, `protected` und `private`.
- Den Modifier `static`. Dieser erzeugt Klassenmethoden und -variablen.
- Den Modifier `final`. Dieser verhindert, daß Klassen abgeleitet und Methoden überschrieben werden können, und er macht Variablen zu Konstanten.
- Den Modifier `abstract`. Er dient der Erstellung abstrakter Klassen und Methoden.
- Die Modifier `synchronized` und `volatile`. Diese werden in Zusammenhang mit Threads verwendet.

Um einen Modifier zu verwenden, integrieren Sie das entsprechende Schlüsselwort in der Definition der Klasse, Methode oder Variablen, auf die Sie diesen anwenden wollen. Der Modifier geht dem Rest der Anweisung voraus, wie das in den folgenden Beispielen gezeigt ist:

```
public class MyApplet extends java.applet.Applet { ... }
```

```
private boolean killJabberwock;
```

```
static final double weeks = 9.5;
```

```
protected static final int MEANINGOFLIFE = 42;
```

```
public static void main(String arguments[]) { ... }
```

Wenn Sie mehr als einen Modifier in einer Anweisung verwenden, können Sie diese in beliebiger Reihenfolge angeben, solange alle Modifier vor dem Element stehen, auf das sie angewendet werden. Stellen Sie sicher, daß Sie den Rückgabetyt einer Methode - z.B. `void` - nicht wie einen der Modifier behandeln.

Modifier sind optional - was Sie daran merken sollten, daß wir in den letzten zwei Wochen nur sehr wenige davon

verwendet haben. Es gibt aber, wie Sie sehen werden, viele gute Gründe, sie zu verwenden.

Zugriffskontrolle für Methoden und Variablen

Mit »Zugriffskontrolle« ist hier die Kontrolle der Sichtbarkeit gemeint. Ist eine Methode oder Variable für eine andere Klasse sichtbar, können ihre Methoden auf diese Methode oder Variable verweisen (sie aufrufen oder modifizieren). Um eine Methode oder Variable vor solchen Referenzen zu »schützen«, können Sie die vier in den nächsten Abschnitten beschriebenen Sichtbarkeitssebenen anwenden. Jede Ebene ist einschränkender als die vorherige und bietet damit mehr Schutz.

Vier Schutzebenen

Die vier Schutzebenen (public, package, protected und private) bezeichnen die grundlegenden Beziehungen, die eine Methode oder Variable einer Klasse mit den anderen Klassen im System haben kann.

Der Standardzugriff

In den meisten Beispielen dieses Buches haben Sie keine bestimmte Zugriffskontrolle angegeben. Variablen und Methoden wurden mit Anweisungen wie den folgenden deklariert:

```
String singer = "Phil Harris";
boolean digThatCrazyBeat() {
    return true;
}
```

Eine Variable oder Methode, die ohne einen Modifier für die Zugriffskontrolle deklariert wird, ist für jede Klasse innerhalb desselben Pakets verfügbar. Sie haben bereits erfahren, daß die Klassen in der Klassenbibliothek von Java in Paketen organisiert sind. Das Paket java.awt ist eines davon - es bietet eine Reihe von Klassen mit Verhaltensweisen für das Abstract Windowing Toolkit von Java.

Jede Variable, die ohne Modifier deklariert wurde, kann von anderen Klassen in demselben Paket gelesen bzw. verändert werden. Jede Methode, die auf diese Art deklariert wurde, kann von jeder anderen Klasse in demselben Paket aufgerufen werden. Keine andere Klasse außerhalb des Pakets kann allerdings auf diese Elemente zugreifen.

Diese Ebene der Zugriffskontrolle kontrolliert allerdings nur wenig beim Zugriff. Wenn Sie beginnen darüber nachzudenken, wie Ihre Klasse von anderen Klassen verwendet wird, werden Sie einen der drei Modifier öfter verwenden, als die standardmäßige Zugriffskontrolle zu akzeptieren.



Die vorherige Diskussion wirft die Frage auf, in welchem Paket sich Ihre eigenen Klassen, die Sie bis zu diesem Zeitpunkt erstellt haben, befanden. Wie Sie später am heutigen Tag sehen werden, können Sie Ihre Klassen zu einem Mitglied eines Pakets machen, indem Sie die Anweisung package verwenden. Wenn Sie diesen Ansatz nicht verwenden, dann wird Ihre Klasse in ein Paket gepackt, in dem sich alle anderen Klassen ohne explizite Paketzugehörigkeit befinden.

private

Die höchste Schutzebene ist das Gegenteil zu public. private-Methoden und -Variablen sind nur innerhalb der eigenen Klasse sichtbar:

```
public class APrivateClass {
    private int aPrivateInt;
    private String aPrivateString;
    private float aPrivateMethod() {
        ...
    }
}
```

Das mag zwar extrem einschränkend erscheinen, ist aber die vorwiegend angewandte Schutzebene. Private Daten, interne Zustände oder eindeutige Darstellungen in Ihrer Implementierung - kurz: alles, was die Subklassen nicht direkt mitnutzen sollen - sind `private`. Bedenken Sie, daß die primäre Aufgabe eines Objekts die Kapselung seiner Daten ist, sie also vor der Welt zu verbergen, damit sie nicht manipuliert werden können. Sie können trotzdem weniger einschränkende Methoden verwenden, jedoch ist ein straffer Zügel über Ihre internen Darstellungen wichtig, wie Sie noch sehen werden. Sie trennen dadurch das Design von der Implementierung, minimieren die Informationsmenge, die eine Klasse von einer anderen braucht, um ihre Aufgabe zu erfüllen, und reduzieren den Umfang der im Code erforderlichen Änderungen, falls Sie die Darstellung ändern.

public

Da jede Klasse eine Insel für sich ist, betrifft diese Beziehung die Unterscheidung zwischen dem internen und externen Bereich einer Klasse. Eine Methode oder Variable ist für die Klasse, in der sie definiert wurde, sichtbar. Was aber muß geschehen, wenn Sie sie für *alle* Klassen außerhalb dieser Klasse sichtbar machen wollen?

Die Antwort ist klar: Sie deklarieren die Methode oder Variable einfach als `public`. Fast jede in diesem Buch definierte Methode und Variable wurde der Einfachheit halber `public` deklariert. Wenn Sie mit Ihrem eigenen Code arbeiten, können Sie den Zugriff weiter einschränken. Nachfolgend einige Beispiele mit `public`-Deklarationen:

```
public class APublicClass {
    public int aPublicInt;
    public String aPublicString;
    public float aPublicMethod() {
        ...
    }
}
```

Eine Variable oder Methode mit `public`-Zugriff ist am stärksten sichtbar, d.h. sie kann von allen gesehen werden und alle können auf sie zugreifen. Selbstverständlich ist das nicht immer wünschenswert. In diesen Fällen greift dann eine der anderen Schutzebenen.

protected

Die dritte Beziehung betrifft eine Klasse und ihre gegenwärtigen und zukünftigen Subklassen. Auf Methoden und Variablen von `protected`-Klassen können nur Subklassen der Klassen zugreifen.

Subklassen stehen einer Superklasse aus folgenden Gründen viel näher als »fremde« Klassen (Klassen anderer Pakete):

- Subklassen kennen normalerweise die Interna einer Superklasse.
- Subklassen werden in der Regel von jemandem geschrieben, der den Quellcode kennt.
- In Subklassen müssen die Daten einer Superklasse häufig geändert oder erweitert werden.

Diese Ebene bietet mehr Schutz und grenzt den Zugriff noch weiter ein, erlaubt den Subklassen aber immer noch vollen Zugriff. Im folgenden ein paar Beispiele für Deklarationen mit `protected`:

```
public class AProtectedClass {
    protected int aProtectedInt = 4;
    protected String aProtectedString = "and a 3 and a ";
    protected float aProtectedMethod() {
        ...
    }
}

public class AProtectedClassSubclass extends AProtectedClass {
    public void testUse() {
        AProtectedClass aPC = new AProtectedClass();
        System.out.println(aPC.aProtectedString + aPC.aProtectedInt);
        aPC.aProtectedMethod(); // Alle hier sind A.O.K.(absolut OK)
    }
}

public class AnyClassInTheSamePackage {
    public void testUse() {
        AProtectedClass aPC = new AProtectedClass();
```

```
System.out.println(aPC.aProtectedString + aPC.aProtectedInt);
aPC.aProtectedMethod(); // Keine hiervon ist legal
}
}
```

Obwohl sich `AnyClassInTheSamePackage` im gleichen Paket befindet wie `AProtectedClass`, ist sie keine Subklasse davon (sondern von `Object`). Nur Subklassen ist es gestattet, `protected`-Variablen und -Methoden zu sehen und zu verwenden.

Eines der deutlichsten Beispiele der Notwendigkeit für diese spezielle Zugriffsebene zeigt sich in der Unterstützung einer `public`-Abstraktion in Ihrer Klasse. Was die Außenwelt betrifft, haben Sie eine einfache `public`-Schnittstelle (über Methoden) für jede Abstraktion, die Sie für Ihre Benutzer definiert haben. Eine komplexere Darstellung und die Implementierung, die davon abhängt, ist im Inneren verborgen. Wenn Subklassen diese Darstellung erweitern und ändern, müssen sie die zugrundeliegende konkrete Darstellung erhalten:

```
public class SortedList {
    protected BinaryTree theBinaryTree;
    ...
    public Object[] theList() {
        return theBinaryTree.asArray();
    }
    public void add(Object o) {
        theBinaryTree.addObject(o);
    }
}
public class InsertSortedList extends SortedList {
    public void insert(Object o, int position) {
        theBinaryTree.insertObject(o, position);
    }
}
```

Ohne in der Lage zu sein, auf `theBinaryTree` direkt zuzugreifen, muß die `insert()`-Methode die Liste als `Object-Array` über die `public`-Methode `theList()` erhalten, ein neues größeres `Array` zuweisen und das neue Objekt manuell einfügen. Da sie »sieht«, daß ihre Superklasse `BinaryTree` verwendet, um die sortierte Liste zu implementieren, kann sie die in `BinaryTree` befindliche Methode `insertObject()` benutzen, um diese Aufgabe zu erfüllen.

Einige Sprachen, z.B. CLU, experimentieren mit expliziteren Formen des Anhebens und Senkens der Abstraktionsebene, um das gleiche Problem auf allgemeinere Art zu lösen. In Java löst `protected` das Problem nur teilweise, indem das Konkrete vom Abstrakten getrennt werden kann. Der Rest wird dem Programmierer überlassen.

Konventionen für den Zugriff auf Instanzvariablen

Als allgemeine Faustregel gilt, daß eine Instanzvariable `private` sein sollte, wenn sie nicht konstant ist (wie das definiert wird, lernen Sie in Kürze). Falls Sie diese Faustregel nicht einhalten, stoßen Sie auf folgendes Problem:

```
public class AFoolishClass {
    public String aUsefulString;
    ... // Den nützlichen Wert für die Zeichenkette einrichten
}
```

Diese Klasse kann `aUsefulString` zur Verwendung durch andere Klassen einrichten, die diese (nur) lesen können. Da sie nicht `private` ist, können sich die anderen Klassen jedoch so verhalten:

```
AFoolishClass aFC = new AFoolishClass();
aFC.aUsefulString = "oops!";
```

Da es keine Möglichkeit gibt, die Schutzebene getrennt zum Lesen und Schreiben von Instanzvariablen zu bestimmen, sollten sie immer `private` sein.

Dem aufmerksamen Leser ist wahrscheinlich nicht entgangen, daß diese Regel in vielen Beispielen dieses Buches nicht eingehalten wird. Der Grund hierfür ist lediglich, die Beispiele übersichtlich und kurz zu halten. (Sie werden bald feststellen, daß viel Platz nötig ist, wenn man das richtigstellt.) Eine Verwendung kann nicht umgangen werden: Die `System.out.print()`-Aufrufe überall im Buch müssen die `public`-Variable `out` direkt benutzen. Sie können diese

final-Systemklasse (die Sie eventuell anders geschrieben haben) nicht ändern. Sie können sich die verheerenden Folgen vorstellen, wenn jemand versehentlich den Inhalt dieser (globalen) public-Variablen ändert!

Vergleich der Zugriffskontrollebenen

Die Unterschiede zwischen den verschiedenen Arten des Zugriffsschutzes können einen sehr schnell verwirren. Speziell im Fall von protected-Methoden und -Variablen. Tabelle 15.1 hilft, die Unterschiede zwischen der am wenigsten einschränkenden (public) bis zur restriktivsten (private) Form des Zugriffsschutzes zu verdeutlichen.

Tabelle 15.1: Die unterschiedlichen Ebenen des Zugriffsschutzes

Sichtbarkeit	public	protected	Default	private
Innerhalb derselben Klasse	Ja	Ja	Ja	Ja
Von einer bel. Klasse im selben Paket	Ja	Ja	Ja	Nein
Von einer bel. Klasse außerhalb des Pakets	Ja	Nein	Nein	Nein
Von einer Subklasse im selben Paket	Ja	Ja	Ja	Nein
Von einer Subklasse außerhalb des Pakets	Ja	Ja	Nein	Nein

Zugriffskontrolle und Vererbung

Ein letztes Thema bei der Zugriffskontrolle für Methoden steht im Zusammenhang mit Subklassen. Wenn Sie eine Subklasse erstellen und eine Methode überschreiben, dann müssen Sie die Zugriffskontrolle der Original-Methode beachten.

Sie werden sich vielleicht erinnern, daß die Applet-Methoden wie z.B. init() und paint() in Ihren eigenen Applets public sein mußten.

Als allgemeine Regel kann man folgendes sagen: Sie können eine Methode in Java nicht überschreiben und der neuen Methode eine stärkere Zugriffskontrolle zuweisen als die Original-Methode hatte. Allerdings haben Sie die Möglichkeit, die Zugriffskontrolle zu lockern. Folgende Regeln gelten für geerbte Methoden:

- Methoden, die in einer Superklasse als public deklariert sind, müssen in allen Subklassen ebenfalls als public deklariert werden (aus diesem Grund sind die meisten der Methoden eines Applets public).
- Methoden, die in einer Superklasse als protected deklariert sind, müssen in einer Subklasse entweder als protected oder als public deklariert sein; private ist nicht möglich.
- Methoden, die ohne Zugriffskontrolle deklariert wurden (es wurde kein Modifier verwendet), können in einer Subklasse mit einer strikteren Zugriffskontrolle versehen werden.
- Methoden, die als private deklariert sind, werden nicht vererbt, so daß diese Regeln nicht greifen.

Accessor-Methoden

Wie kann die Außenwelt auf private-Instanzvariablen zugreifen? Indem »Accessor«- Methoden geschrieben werden:

```
public class ACorrectClass {
    private String aUsefulString;
    public String aUsefulString() { //Wert holen
        return aUsefulString;
    }
    protected void aUsefulString(String s) { //Wert setzen
        aUsefulString = s;
    }
}
```

Die Verwendung von Methoden für den Zugriff auf eine Instanzvariable ist die häufigste Vorgehensweise in objektorientierten Programmen. Diese Vorgehensweise in allen Klassen zahlt sich aus, da die Programme robuster werden und gut wiederverwendet werden können.

Eine Namenskonvention für Accessor-Methoden ist das Voranstellen des Präfixes `get` bzw. `set` vor den Variablennamen. Diese Variante der Namensgebung erhöht die Lesbarkeit des Codes und Sie laufen nicht Gefahr, irgendwann einmal den Code ändern zu müssen, falls die andere Variante nicht mehr zulässig ist. Davon abgesehen ist es eine Frage des persönlichen Stils, welche Konvention Sie verwenden. Wenn Sie sich für eine entschieden haben, sollten Sie diese allerdings konsequent verwenden.

```
Class Circle
private int x, y, radius;

public int getRadius(){
return Radius
}
public int setRadius(int value){
radius = value
draw();
doOtherStuff();
return Radius
}
```

Aufruf:

```
oldRadius = theCircle.getRadius(); // Den Wert ermitteln
newRadius = theCircle.setRadius(4); // Den Wert setzen usw.
```



Diese Konvention wird mit jeder Version von Java immer mehr zum Standard. Sie werden sich vielleicht daran erinnern, daß die Methode `size()` der Klasse `Dimension` mit Java 1.2 in `getSize()` umbenannt wurde. Sie werden diese Namenskonventionen vielleicht auch für Ihre eigenen Accessor-Methoden verwenden wollen, um Klassen verständlicher zu machen.

Klassenvariablen und -methoden

Was muß geschehen, wenn Sie eine Variable erstellen möchten, die alle Instanzen einer Klasse sehen und verwenden soll? Jede Instanz einer Instanzvariablen hat eine eigene Kopie der Variablen, so daß ihr Sinn zunichte gemacht werden würde. Wenn Sie sie in die Klasse setzen, gibt es nur eine Kopie und alle Instanzen der Klasse nutzen sie gemeinsam. Das nennt man *Klassenvariable*:

```
public class Circle {
public static float pi = 3.14159265F;
public float area(float r) {
return pi * r * r;
}
}
```

Aufgrund historischer Verflechtungen nutzt Java das Wort `static`, um Klassenvariablen und -methoden zu deklarieren. Wann immer Sie das Wort `static` sehen, denken Sie daran, sich geistig »Klasse« vorzustellen.

Instanzen können auf ihre eigenen Klassenvariablen so verweisen, als wären es Instanzvariablen, wie Sie im letzten Beispiel gesehen haben. Da `pi` public ist, können auch Methoden anderer Klassen darauf verweisen:

```
float circumference = 2 * Circle.pi * r;
```

Auch Instanzen von `Circle` können diese Zugriffsform benutzen. In den meisten Fällen ist das der Klarheit halber die bevorzugte Form, auch für Instanzen. Sie zeigt dem Leser sofort auf, daß und wo eine Klassenvariable benutzt wird und daß sie global in allen Instanzen vorkommt. Das mag pedantisch erscheinen, macht aber alles viel übersichtlicher.



Nebenbei bemerkt, falls Sie irgendwann über den Zugriff auf eine Klassenvariable Ihre Meinung ändern, sollten Sie für die Instanz (oder sogar die Klasse) Accessor-Methoden erstellen, um sie vor solchen Änderungen zu schützen.

Klassenmethoden werden analog definiert. Auf sie können Instanzen ihrer Klasse genauso zugreifen, während Instanzen anderer Klassen nur mit dem vollen Klassennamen auf sie zugreifen können. Im folgenden Beispiel (Listing 15.1) definiert eine Klasse Klassenmethoden, um ihre eigenen Instanzen zu zählen:

Listing 15.1: Der gesamte Quelltext von CountInstances.java

```

1: public class CountInstances {
2:     private static int numInstances = 0;
3:
4:     protected static int getNumInstances() {
5:         return numInstances;
6:     }
7:
8:     private static void addInstance() {
9:         numInstances++;
10:    }
11:
12:    CountInstances() {
13:        CountInstances.addInstance();
14:    }
15:
16:    public static void main(String arguments[]) {
17:        System.out.println("Starting with " +
18:            CountInstances.getNumInstances() + " instances");
19:        for (int i = 0; i < 10; ++i)
20:            new CountInstances();
21:        System.out.println("Created " +
22:            CountInstances.getNumInstances() + " instances");
23:    }
24: }
```

Das Programm erzeugt die folgende Ausgabe:

```

Started with 0 instances
Creates 10 instances
```

Dieses Beispiel hat eine ganze Reihe von Features. Sie sollten sich die Zeit nehmen, es Zeile für Zeile durchzuarbeiten. In Zeile 2 deklarieren Sie eine private-Klassenvariable (numInstances), die die Anzahl der Instanzen speichert. Es wird eine Klassenvariable (die Variable ist als static deklariert) verwendet, da die Anzahl der Instanzen für die Klasse als Gesamtes relevant ist und nicht für die einzelnen Instanzen. Sie ist außerdem private, so daß sie denselben Regeln bezüglich der Accessor-Methoden genügt wie Instanzvariablen.

Beachten Sie bitte, daß numInstances in derselben Zeile mit 0 initialisiert wird. Genauso wie eine Instanzvariable initialisiert wird, wenn deren Instanz erzeugt wird, wird eine Klassenvariable initialisiert, wenn deren Klasse erzeugt wird. Die Initialisierung einer Klasse findet statt, bevor irgend etwas anderes mit der Klasse oder deren Instanzen geschehen kann. Aus diesem Grund wird das Beispiel wie geplant funktionieren.

In den Zeilen 4-6 erstellen Sie eine get-Methode (getNumInstances()) für die private -Klassenvariable, um deren Wert auszulesen. Diese Methode ist ebenfalls als Klassenmethode deklariert, da diese zu der Klassenvariablen gehört. Die Methode getNumInstances() ist als protected und nicht als public deklariert, da nur diese Klasse und vielleicht noch Subklassen davon an dem Wert interessiert sind. Andere Klassen bleiben aus diesem Grund außen vor.

Beachten Sie bitte, daß es keine Accessor-Methode zum Setzen des Werts gibt. Der Grund dafür ist, daß der Wert der Variablen nur dann inkrementiert werden soll, wenn eine neue Instanz erzeugt wird. Sie sollte nicht einfach so auf einen Wert gesetzt werden. Deshalb erstellen Sie anstelle einer Accessor-Methode eine spezielle private- Methode mit dem Namen addInstance() in den Zeilen 8-10, die den Wert von numInstances um 1 inkrementiert.

In den Zeilen 12-14 befindet sich der Konstruktor dieser Klasse. Erinnern Sie sich bitte daran, daß Konstruktoren aufgerufen werden, sobald ein neues Objekt erzeugt wird. Dies stellt den sinnvollsten Ort dar, um die Methode addInstance() aufzurufen und die Variable zu inkrementieren.

Schließlich deutet die `main()`-Methode darauf hin, daß Sie dieses Programm als Java- Applikation ausführen können. Alle anderen Methoden können Sie mit dieser testen. In der `main()`-Methode erzeugen Sie 10 Instanzen der Klasse `CountInstances`. Anschließend wird der Wert der Klassenvariablen `numInstances` ausgegeben (der 10 sein sollte).

Der final-Modifizier

Der `final`-Modifizier ist sehr vielseitig:

- Wird der `final`-Modifizier auf eine Variable angewandt, bedeutet das, daß die Variable konstant ist.
- Wird der `final`-Modifizier auf eine Methode angewandt, bedeutet das, daß die Methode von Subklassen nicht überschrieben werden kann.
- Wird der `final`-Modifizier auf eine Klasse angewandt, bedeutet das, daß von der Klasse keine Subklassen erstellt werden können.

final-Variablen

Um Konstanten in Java zu deklarieren, verwenden Sie `final`-Variablen:

```
public class AnotherFinalClass {
public static final int aConstantInt = 123;
public final String aConstantString = "Hello World!";
}
```

`final`-Klassen und -Instanzvariablen können in Ausdrücken wie normale Klassen und Instanzvariablen verwendet, aber nicht geändert werden. Deshalb muß `final`-Variablen ihr (konstanter) Wert zum Zeitpunkt der Deklaration zugewiesen werden. Klassen können über `final`-Klassenvariablen anderen Klassen nützliche Konstanten liefern. Andere Klassen greifen auf sie wie oben zu: `AnotherFinalClass.aConstantInt`.

Lokale Variablen (diejenigen, die in Codeblöcken zwischen Klammern stehen, z.B. in `while`- oder `for`-Schleifen) können unter Java 1.02 nicht `final` deklariert werden. Vor lokalen Variablen dürfen überhaupt keine Modifizier stehen:

```
{
int aLocalVariable; // Ich komme ganz gut ohne Modifizier zurecht...
...
}
```

In Java 1.2 wurde dies im Zuge der Einführung der Inner Classes möglich.

final-Methoden

Nachfolgend ein Beispiel mit `final`-Methoden:

```
public class MyPenultimateFinalClass {
public static final void aUniqueAndReallyUsefulMethod() {
...
}
public final void noOneGetsToDoThisButMe() {
...
}
}
```

`final`-Methoden können nicht in Subklassen überschrieben werden. Eine Methode soll nicht das letzte Wort in einer Implementierung haben, weshalb sollte auf Methoden also dieser Modifizier angewandt werden?

Aus Gründen der Effizienz. Wenn Sie eine Methode `final` deklarieren, kann der Compiler davon ausgehen, daß nie eine Subklasse davon auftaucht und daß die Definition der Methode nicht geändert werden kann.

Die Java-Klassenbibliothek deklariert viele übliche Methoden `final`, so daß Sie Vorteile in bezug auf Geschwindigkeit haben. Im Fall von Klassen, die bereits `final` sind, ist das absolut sinnvoll. Die wenigen `final`-Methoden, die in Nicht-`final`-Klassen deklariert sind, sind eher ein Ärgernis. Sie können sie nicht in Subklassen überschreiben. Ist Effizienz in künftigen Java-Versionen keine vorrangige Frage mehr, werden eventuell viele dieser `final`-Methoden wieder »aufgetaut«, so daß entgangene Flexibilität des Systems wiederhergestellt wird.



private-Methoden sind effektiv final, wie das auch bei allen Methoden in einer final -Klasse der Fall ist. Die Kennzeichnung dieser Methoden mit final (was die Java- Bibliothek manchmal macht) ist zulässig, aber redundant. Der derzeitige Compiler behandelt sie ohnehin als final. final-Methoden können aus den gleichen Sicherheitsgründen wie final-Klassen benutzt werden, jedoch ist das eher selten.

Falls Sie (wie empfohlen) reichlich Gebrauch von Accessor-Methoden machen und sich um die Effizienz sorgen, sehen Sie sich diese neue Fassung von ACorrectClass an, die viel schneller ist:

```
public class ACorrectFinalClass {
    private String aUsefulString;
    public final String aUsefulString() { // Läuft jetzt schneller
        return aUsefulString;
    }
    protected final void aUsefulString(String s) { // Auch schneller
        aUsefulString = s;
    }
}
```

Künftige Java-Compiler werden sicherlich klug genug sein, um einfache Methoden automatisch zu verarbeiten, deshalb müssen Sie final in solchen Fällen eventuell nicht mehr verwenden.

final-Klassen

Nachfolgend die Deklaration einer final-Klasse:

```
public final class AFinalClass {
    ...
}
```

Eine Klasse wird aus zwei Gründen final deklariert: erstens wegen der Sicherheit. Niemand außer Ihnen soll in der Lage sein, Subklassen und neue oder andere Instanzen davon zu erstellen. Zweitens wegen der Effizienz. Sie möchten sich darauf verlassen können, daß sich Instanzen in nur einer Klasse (nicht in Subklassen) befinden, so daß Sie sie optimieren können.

In der Java-Klassenbibliothek werden final-Klassen reichlich verwendet. Beispiele des ersten Grundes für final sind folgende Klassen: java.lang.System sowie InetAddress und Socket aus dem Paket java.net. Ein gutes Beispiel für den zweiten Grund von final ist java.lang.String.

Sie werden zwar selten Gelegenheit haben, eine final-Klasse selbst zu erstellen, jedoch erhalten Sie reichlich Gelegenheit, sich darüber zu ärgern, daß bestimmte Systemklassen final sind (und damit ihre Erweiterung erschweren). Nehmen wir das für mehr Sicherheit und Effizienz eben in Kauf. Wir wollen hoffen, daß Effizienz bald kein Thema mehr ist und einige dieser Klassen wieder public sein werden.

abstract-Methoden und -Klassen

Bei der Anordnung von Klassen in einer Vererbungshierarchie geht man von der Annahme aus, daß die höheren Klassen abstrakter und allgemeiner sind, während die unteren Subklassen konkreter und spezifischer sind. Meist verwendet man bei der Auslegung von Klassen gemeinsame Design- und Implementierungsmerkmale in einer Superklasse. Ist dieser gemeinsame Speicherort der primäre Grund, daß eine Superklasse existiert und sollen nur Subklassen verwendet werden, nennt man eine solche Superklasse eine abstrakte Klasse. Abstrakte Klassen werden mit dem Modifier abstract deklariert.

Von abstract-Klassen können keine Instanzen erstellt werden, jedoch können sie alles enthalten, was in einer normalen Klasse stehen kann. Darüber hinaus sind Präfixe für Methoden mit dem Modifier abstract zulässig. Nichtabstrakte Klassen dürfen diesen Modifier nicht verwenden. Hier ein Beispiel:

```
public abstract class MyFirstAbstractClass {
    int anInstanceVariable;
    public abstract int aMethodMyNonAbstractSubclassesMustImplement();
}
```

```

public void doSomething() {
... // Eine normale Methode
}
}
public class AConcreteSubClass extends MyFirstAbstractClass {
public int aMethodMyNonAbstractSubclassesMustImplement() {
... // Wir müssen diese Methode implementieren
}
}
}

```

Und hier ein paar Versuche, diese Klassen zu benutzen:

```

Object a = new MyFirstAbstractClass(); // Unzulässig, ist abstrakt
Object c = new AConcreteSubClass();    // OK, das ist eine konkrete Subklasse

```

abstract-Methoden brauchen keine Implementierung, während das bei nichtabstrakten Subklassen notwendig ist. Die abstract-Klasse stellt nur eine Maske für die Methoden bereit, die später von anderen implementiert werden. In der Java-Klassenbibliothek gibt es viele abstract-Klassen, für die es im System keine dokumentierten Subklassen gibt. Sie dienen lediglich als Grundlage zum Erstellen von Subklassen in eigenen Programmen.

Die Verwendung einer abstract-Klasse zur Umsetzung eines reinen Designs, d.h. mit nichts als abstract-Methoden, wird in Java mit einer Schnittstelle (wird morgen behandelt) besser erreicht. Ruft ein Design eine Abstraktion auf, die einen Instanzzustand und/oder eine teilweise Implementierung beinhaltet, ist eine abstrakte Klasse allerdings nicht die einzige Wahl. In älteren objektorientierten Sprachen sind abstrakte Klassen lediglich eine Konvention. Sie haben sich als derart nützlich erwiesen, daß sie in Java nicht nur in der hier beschriebenen Form, sondern auch in der reineren Form von Schnittstellen unterstützt werden.

Was sind Pakete?

Pakete sind, wie bereits einige Male erwähnt, eine Möglichkeit, Gruppen von Klassen zu organisieren. Ein Paket enthält eine beliebige Anzahl von Klassen, die jeweils nach Sinn, Verwendung oder auf der Grundlage der Vererbung zusammengefaßt werden.

Wozu sind Pakete notwendig? Wenn Ihre Programme klein sind und nur eine beschränkte Anzahl von Klassen verwenden, fragen Sie sich eventuell, warum Sie sich überhaupt mit Paketen befassen sollen. Aber je mehr Java-Programmierungen Sie vornehmen, desto mehr Klassen werden Sie verwenden. Und obwohl diese Klassen im einzelnen ein gutes Design aufweisen, einfach wiederzuverwenden sind, eingekapselt sind und über spezielle Schnittstellen zu anderen Klassen verfügen, stehen Sie vor der Notwendigkeit, eine größere Organisationseinheit zu verwenden, die es ermöglicht, Ihre Pakete zu gruppieren.

Pakete sind aus den folgenden Gründen sinnvoll:

- Sie ermöglichen eine Organisation der Klassen in Einheiten. Ebenso wie Sie Ordner oder Verzeichnisse auf der Festplatte für die Verwaltung Ihrer Dateien und Anwendungen anlegen, können Sie mit Hilfe von Paketen die Klassen in Gruppen verwalten und damit nur jene Elemente verwenden, die Sie für ein Programm benötigen.
- Pakete reduzieren das Problem der Namenskonflikte. Je größer die Anzahl von Java-Klassen wird, desto wahrscheinlicher wird es, daß Sie denselben Namen für eine Klasse verwenden wie jemand anders. Damit ist die Möglichkeit von Namenskonflikten und fehlerhaften Ergebnissen relativ hoch, wenn Sie die Klassengruppen in einem einzelnen Programm integrieren. Mit Paketen können Sie Klassen desselben Namens verbergen und damit Konflikte vermeiden.
- Pakete dienen dazu, Klassen, Variablen und Methoden in größerem Umfang zu schützen, als dies auf der Basis von Klassen allein möglich ist. Über den Schutz durch Pakete erfahren Sie später in dieser Lektion noch genaueres.
- Pakete lassen sich zur Identifikation von Klassen verwenden. Wenn Sie zum Beispiel für die Durchführung eines bestimmten Zwecks einen Satz von Klassen implementiert haben, könnten Sie ein Paket mit diesen Klassen durch einen eindeutigen Namen kennzeichnen, der Sie oder Ihre Organisation wiedergibt.

Obwohl ein Paket im allgemeinen aus einer Sammlung von Klassen besteht, können Pakete wiederum auch andere Pakete enthalten und damit eine Hierarchieform bilden, die der Vererbungshierarchie nicht unähnlich ist. Jede »Ebene« stellt dabei meist eine kleinere und noch spezifischere Gruppe von Klassen dar. Die Java-Klassenbibliothek selbst ist anhand dieser Struktur definiert. Die oberste Ebene trägt den Namen java; die nächste Ebene enthält Namen wie io, net, util und awt, die letzte und niedrigste Ebene enthält dann z.B. das Paket image.



Nach der geltenden Konvention gibt die erste Ebene der Hierarchie den (global eindeutigen) Namen der Firma, die das bzw. die Java-Pakete entwickelt hat, an. Die Klassen von SUN Microsystems beispielsweise, die nicht Teil der Java-Standardumgebung sind, beginnen alle mit dem Präfix `sun`. Klassen, die Netscape zusammen mit der Implementation einfügt, sind im Paket `netscape` enthalten. Das Standardpaket `java` bildet eine Ausnahme von dieser Regel, da es so grundlegend ist und eventuell eines Tages auch von vielen anderen Firmen implementiert wird. Nähere Informationen zu den Namenskonventionen bei Paketen erhalten Sie später, wenn Sie eigene Pakete erstellen.

Pakete verwenden

Sie haben in diesem Buch bereits mehrfach Pakete verwendet. Jedesmal, wenn Sie den Befehl `import` benutzt haben, und immer dann, wenn Sie einen Bezug zu einer Klasse anhand des kompletten Paketnamens (z.B. `java.awt.Color`) hergestellt haben, haben Sie ein Paket verwendet. Im folgenden erfahren Sie, wie Sie Klassen aus anderen Paketen in eigenen Programmen benutzen können. Damit soll dieses Thema vertieft und sichergestellt werden, daß Sie es verstanden haben.

Um eine Klasse zu verwenden, die in einem Paket enthalten ist, können Sie eine der drei folgenden Techniken verwenden:

- Wenn sich die gewünschte Klasse in `java.lang` (z.B. `System` oder `Date`) befindet, können Sie diese Klasse einfach benutzen, indem Sie auf den Namen dieser Klasse Bezug nehmen.
- Wenn sich die gewünschte Klasse in einem anderen Paket befindet, können Sie auf diese anhand des kompletten Namens, einschließlich des Paketnamens (z.B. `java.awt.Font`) Bezug nehmen.
- Bei häufig verwendeten Klassen aus anderen Paketen können Sie einzelne Klassen oder auch das gesamte Klassenpaket importieren. Sobald eine Klasse oder ein Paket importiert wurde, können Sie darauf Bezug nehmen, indem Sie den Namen der Klasse verwenden.

Was ist mit Ihren eigenen Klassen in Ihren Programmen, die nicht zu irgendeinem Paket gehören? Die Regel besagt, daß eine nicht exakt für ein bestimmtes Paket definierte Klasse in einem unbenannten Standardpaket plaziert wird. Den Bezug zu diesen Klassen stellen Sie her, indem Sie den Klassennamen an einer beliebigen Position im Code angeben.

Komplette Paket- und Klassennamen

Um den Bezug zur Klasse in einem anderen Paket herzustellen, können Sie dessen kompletten Namen verwenden: Der Klassenname steht vor den Paketnamen. Sie müssen die Klassen oder Pakete nicht importieren, um sie auf diese Art zu verwenden.

```
java.awt.Font f=new.java.awt.Font()
```

Wenn Sie eine Klasse in einem Programm nur ein- oder zweimal verwenden, sollten Sie den kompletten Namen angeben. Wenn Sie eine bestimmte Klasse jedoch häufig benötigen oder der Paketname selbst wirklich lang ist und viele Unterpakete enthält, lohnt es sich, diese Klasse zu importieren, um Zeit bei der Eingabe des Namens zu sparen.

Der Befehl import

Sie können Klassen mit dem Befehl `import` importieren, wie Sie dies in den Beispielen dieses Buches bereits durchgeführt haben. Mit der folgenden Eingabe importieren Sie eine einzelne Klasse:

```
import java.util.Vector;
```

oder Sie importieren ein komplettes Klassenpaket, indem Sie einen Asterisk (*) anstelle der einzelnen Klassennamen verwenden:

```
import java.awt.*
```



Um technisch korrekt zu sein: Dieser Befehl importiert nicht alle Klassen in einem Paket - er importiert nur jene Klassen, die mit `public` als öffentlich erklärt wurden und selbst hierbei werden nur jene Klassen importiert, auf welche sich der Code selbst bezieht. Zu diesem Thema erhalten Sie später in dieser Lektion weitere Informationen.

Beachten Sie, daß der Asterisk (*) in diesem Beispiel nicht, wie Sie das eventuell gewohnt sind, in der Befehlszeile verwendet wird, um die Inhalte eines Verzeichnisses zu definieren oder mehrere Dateien anzugeben. Wenn Sie z.B. den Inhalt des Verzeichnisses `classes/java/awt/*` auflisten lassen möchten, enthält diese Liste alle Dateien und Unterverzeichnisse, wie `image` und `peer`, in diesem Verzeichnis. Wenn Sie `import java.awt.*` schreiben, werden alle öffentlichen Klassen in diesem Paket importiert, aber keine Unterpakete wie `image` und `peer`. Um alle Klassen in einer komplexen Pakethierarchie zu importieren, müssen Sie jede Ebene dieser Hierarchie explizit manuell importieren. Sie können ferner keine partiellen Klassennamen angeben (z.B. `L*`, um alle Klassen zu importieren, die mit dem Buchstaben `L` beginnen). Entweder importieren Sie alle Klassen in einem Paket oder eine einzelne Klasse.

Die `import`-Anweisungen in Ihrer Klassendefinition sollten am Anfang der Datei stehen, vor allen Klassendefinitionen (aber nach der Paketdefinition - siehe den nächsten Abschnitt).

Empfiehl es sich, die Klassen einzeln zu importieren oder sollten diese besser als Gruppe importiert werden? Dies hängt davon ab, wie speziell Sie verfahren möchten. Wenn Sie eine Gruppe von Klassen importieren, wird dadurch das Programm nicht verlangsamt oder »aufgebläht«; es werden nur diejenigen Klassen geladen, die aktuell vom Code verwendet werden. Wenn Sie Pakete importieren, ist das Lesen des Codes für andere allerdings etwas komplizierter, denn es liegt dann nicht mehr auf der Hand, woher die Klassen stammen. Ob Sie die Klassen einzeln oder als Pakete importieren ist überwiegend eine Frage des eigenen Programmierstils.



Der `import`-Befehl in Java ist dem Befehl `#include` in keiner Weise ähnlich. Daraus ergibt sich ein enormer Code, der über deutlich mehr Zeilen verfügt als das Originalprogramm aufwies. Der `import`-Befehl von Java agiert mehr als eine Art Verbindungsstück. Damit wird dem Java-Compiler und dem Interpreter mitgeteilt, wo (in welchen Dateien) die Klassen, Variablen, Methodennamen und die Methodendefinitionen zu finden sind. Der Umfang einer Klasse wird dadurch nicht erweitert.

Namenskonflikte

Nachdem Sie eine Klasse oder ein Paket von Klassen importiert haben, können Sie sich im allgemeinen auf eine Klasse einfach dadurch beziehen, indem Sie den Namen ohne Paket-Identifikation angeben. Ich sage »im allgemeinen«, weil es einen Fall gibt, der ausdrücklicher definiert werden muß: wenn mehrere Klassen desselben Namens in unterschiedlichen Paketen vorhanden sind.

Im folgenden finden Sie ein Beispiel. Angenommen, Sie importieren die Klassen aus zwei Paketen von den beiden verschiedenen Programmierern (Joe und Eleanor):

```
import joesclasses.*;
import eleanorsclasses.*;
```

Innerhalb von Joes Paket befindet sich eine Klasse `Name`. Leider enthält auch Eleanors Paket eine Klasse mit dem Namen `Name`, die eine komplett andere Bedeutung und Implementation hat. Ein Mensch würde fragen, auf welche Version der `Name`-Klasse sich Ihr Programm bezieht, wenn Sie folgendes eingeben:

```
Name myName = new Name("Susan");
```

Doch dies ist bei Java nicht der Fall. Der Java-Compiler würde sich über den Namenskonflikt beschweren und die Kompilierung des Programms verweigern. In diesem Fall müssen Sie, trotz der Tatsache, daß Sie beide Klassen importiert haben, einen Bezug zur betreffenden `Name`-Klasse anhand des vollständigen Paketnamens einfügen:

```
Name myName = new joesclasses.Name("Susan");
```

Anmerkung zu CLASSPATH und zur Position von Klassen

Ehe ich erkläre, wie Sie eigene Klassenpakete erstellen, möchte ich eine Anmerkung darüber machen, wie Java Pakete und Klassen findet, wenn es Ihre Klassen kompiliert und ausführt.

Damit Java eine Klasse verwenden kann, muß es diese im Dateisystem finden können. Andernfalls erhalten Sie eine Fehlermeldung, die besagt, daß die Klasse nicht existiert. Java verwendet zwei Elemente, um eine Klasse zu finden: den Namen des Pakets selbst und die Verzeichnisse, die in der Variablen `CLASSPATH` aufgelistet sind.

Zunächst zu den Paketnamen. Paketnamen entsprechen den Verzeichnisnamen im Dateisystem, d.h. die Klasse `java.applet.Applet` ist im Verzeichnis `applet` zu finden, welches wiederum im Verzeichnis `java` liegt (also `java/applet/Applet.class`).

Java sucht nach jenen Verzeichnissen innerhalb derjenigen Verzeichnisse, die in der Variablen `CLASSPATH` aufgelistet sind. Wenn Sie sich an den 1. Tag erinnern, als Sie JDK installiert haben, wissen Sie noch, daß Sie eine Variable `CLASSPATH` eingerichtet haben, um auf die verschiedenen Positionen zu verweisen, an denen sich die Java- Klassen befinden. `CLASSPATH` verweist im allgemeinen auf das Verzeichnis `java/lib` in Ihrer JDK-Version, ein Klassenverzeichnis in Ihrer Entwicklungsumgebung (falls vorhanden), eventuell einige Browser spezifischer Klassen und auf das aktuelle Verzeichnis. Wenn Java nach einer Klasse sucht, auf die Sie in der Quelle Bezug genommen haben, wird nach dem Paket- und Klassennamen in einem dieser Verzeichnisse gesucht und eine Fehlermeldung ausgegeben, falls die Klassendatei nicht gefunden werden kann. Die meisten Fehlermeldungen für nicht ladbare Klassendateien werden durch nicht vorhandene `CLASSPATH` -Variablen erzeugt.

Eigene Pakete erstellen

Das Erstellen eigener Pakete ist in Java nicht komplexer als das Erstellen einer Klasse. Um ein Paket mit Klassen zu erstellen, müssen Sie drei grundlegende Schritte ausführen, die in den folgenden Abschnitten erläutert werden.

Paketnamen wählen

Der erste Schritt besteht darin, zu entscheiden, welchen Namen das Paket erhalten soll. Welcher Name für ein Paket gewählt werden soll, hängt davon ab, wie Sie die darin befindlichen Klassen verwenden möchten. Eventuell möchten Sie dem Paket Ihren eigenen Namen geben, oder dieses nach einem bestimmten Teil des Java-Systems benennen, an dem Sie gearbeitet haben (z.B. `graphics` oder `hardware-interfaces`). Wenn Sie beabsichtigen, Ihr Paket im Netz weit zu verbreiten oder als Teil eines kommerziellen Produkts zu vertreiben, sollten Sie einen Paketnamen wählen (oder einen Satz von Paketnamen), der sowohl Sie als auch Ihre Organisation in einmaliger Weise kennzeichnet.

Eine Konvention für die Benennung von Paketen, die von SUN empfohlen wurde, ist, die Elemente des Internet-Domain-Namens zu vertauschen. Wenn SUN also seinem eigenen Rat folgen würde, müßten deren Pakete den Namen `com.sun.java` anstatt nur `java` verwenden. Wenn Ihr Internet-Domain-Name `fooblitzky.eng.nonsense.edu` lautet, könnte der Paketname sein: `edu.nonsense.eng.fooblitzky` (und Sie könnten daran noch weitere Paketnamen anhängen, die sich auf das Produkt oder Sie selbst beziehen).

Die Grundidee ist, daß ein Paketname eindeutig sein sollte. Obwohl Pakete Klassen verbergen können, deren Namen in Konflikt geraten, ist dies auch schon der letzte Schutzmechanismus. Es gibt keine Möglichkeit sicherzustellen, daß das Paket mit dem Paket einer anderen Person in Konflikt gerät, die eventuell denselben Paketnamen verwendet.

Paketnamen beginnen laut Konvention mit einem kleingeschriebenen Buchstaben, um diese von Klassennamen zu unterscheiden. Im kompletten Namen der vordefinierten String-Klasse, `java.lang.String`, ist der Paketname visuell einfach vom Klassennamen zu unterscheiden. Diese Konvention trägt dazu bei, Namenskonflikte zu reduzieren.

Verzeichnisstruktur definieren

Der zweite Schritt für das Erstellen von Paketen besteht darin, eine Verzeichnisstruktur auf Ihrem Datenträger zu erstellen, die dem Paketnamen entspricht. Wenn das Paket nur einen Namen (`mypackage`) enthält, müssen Sie für diesen Namen ein Verzeichnis erstellen. Für das Beispiel des Paketnamens `edu.nonsense.eng.fooblitzky` müssen Sie das Verzeichnis `edu` erstellen, ein Verzeichnis `nonsense` innerhalb von `edu`, ein Verzeichnis `eng` innerhalb von `nonsense` und ein Verzeichnis `fooblitzky` innerhalb von `eng`. Die Klassen- und Quelldateien können dann in das Verzeichnis `fooblitzky` eingefügt werden.

Mit package Klassen in ein Paket einfügen

Der letzte Schritt besteht darin, die Klasse in die Pakete einzufügen; dies geschieht mit dem Befehl `package` in den Quelldateien. Der Befehl `package` sagt: »Diese Klasse soll in diesem Paket plaziert werden«. Er wird wie folgt verwendet:

```
package myclasses;
package edu.nonsense.eng.fooblitzky;
package java.awt;
```

Ein einzelner package-Befehl muß in die erste Zeile des Codes der Quelldatei eingefügt werden, nach den Kommentaren oder Leerzeilen und vor den import-Befehlen.

Wie bereits erwähnt, befindet sich eine Klasse, falls diese nicht über einen package- Befehl verfügt, im Standardpaket und läßt sich von anderen Klassen verwenden. Wenn Sie jedoch einmal damit begonnen haben, Pakete zu verwenden, sollten Sie sicherstellen, daß alle Klassen zu einem Paket gehören, um Verwirrungen über die Zugehörigkeit von Klassen zu vermeiden.

Pakete und Klassenschutz

Gestern haben Sie alles über Schutztechniken erfahren und darüber, wie diese den Methoden und Variablen zugeordnet sind bzw. Sie haben ihre Beziehung zu anderen Klassen kennengelernt. Wenn Sie sich auf Klassen und deren Beziehung zu anderen Klassen in einem Paket beziehen möchten, müssen Sie nur folgende zwei Elemente im Auge behalten: package und public.

Standardmäßig verfügen Klassen über einen Paketschutz, d.h., daß die Klasse auch allen anderen Klassen in diesem Paket zur Verfügung steht, aber außerhalb und von Subpaketen nicht zu sehen oder verfügbar ist. Sie läßt sich nicht anhand des Namens importieren oder für einen Bezug verwenden.

Der Paketschutz findet statt, wenn Sie eine Klasse wie gewöhnlich definieren:

```
class TheHiddenClass extends AnotherHiddenClass {
...
}
```

Um eine Klasse auch außerhalb des betreffenden Pakets zur Verfügung zu stellen, können Sie diese mit einem öffentlichen Schutz versehen, indem Sie public in deren Definition einfügen:

```
public class TheVisibleClass {
...
}
```

Klassen, die mit public definiert sind, lassen sich von anderen Klassen außerhalb des Pakets importieren.

Beachten Sie, daß bei der Verwendung einer import-Anweisung mit einem Asterisk lediglich die öffentlichen Klassen aus diesem Paket importiert werden. Verborgene Klassen bleiben verborgen und können nur von Klassen innerhalb dieses Pakets verwendet werden.

Warum soll eine Klasse in einem Paket verborgen werden? Aus demselben Grund, aus dem Sie auch Variablen und Methoden innerhalb einer Klasse verbergen: damit Sie Hilfsklassen und Verhalten zur Verfügung haben, die ausschließlich für die Implementierung notwendig sind. Damit läßt sich die Schnittstelle Ihres Programms auf die notwendigen Änderungen beschränken. Wenn Sie Ihre Klassen entwerfen, sollten Sie das gesamte Paket im Blick haben und entscheiden, welche Klasse public deklariert werden und welche Klasse verborgen sein soll.

Listing 15.2 zeigt zwei Klassen, die diesen Punkt darstellen. Die erste ist eine öffentliche Klasse, die eine verkettete Liste implementiert, die zweite ist ein Knoten dieser Liste.

Listing 15.2: Der gesamte Quelltext von LinkedList.java

```
1: package  collections;
2:
3: public class  LinkedList {
4:     private Node  root;
5:
6:     public  void  add(Object o) {
7:         root = new Node(o, root);
8:     }
9:     // ...
10: }
11:
12: class  Node {    // nicht public
13:     private Object  contents;
14:     private Node    next;
```

```

15:
16:     Node(Object o, Node n) {
17:         contents = o;
18:         next      = n;
19:     }
20:     // ...
21: }

```



Beachten Sie, daß ich hier zwei Klassendefinitionen in eine Datei eingefügt habe. Ich habe es bereits einmal erwähnt, aber es soll auch an dieser Stelle noch einmal gesagt werden: Sie können in eine Datei beliebig viele Klassendefinitionen einfügen, von diesen kann aber nur eine public deklariert werden. Und dieser Dateiname muß denselben Namen haben wie die öffentliche Klasse. Wenn Java die Datei kompiliert, wird für jede Klassendefinition innerhalb der Datei eine eigene .class-Datei erstellt. In der Realität ist die Eins-zu-Eins-Entsprechung von Klassendefinition zu Datei einfach zu handhaben, weil Sie nicht lange nach der Definition einer Klasse suchen müssen.

Mit der öffentlichen LinkedList-Klasse soll eine Reihe nützlicher public-Methoden (z.B. add()) für andere Klassen bereitgestellt werden. Diese anderen Klassen benötigen keine Informationen über andere Hilfsklassen, die LinkedList verwendet. Node, das eine dieser Hilfsklassen ist, wird deshalb ohne einen public-Modifier deklariert und erscheint nicht als Teil der öffentlichen Schnittstelle des collections-Pakets.



Weil Node nicht public ist, bedeutet dies nicht, daß LinkedList keinen Zugang dazu hat, sobald es in eine andere Klasse importiert ist. Ein Schutz verbirgt nie die gesamten Klassen, sondern dient zur Prüfung der Erlaubnis, ob eine bestimmte Klasse andere Klassen, Variablen und Methoden verwenden kann. Wenn Sie LinkedList importieren und verwenden, wird auch die Node-Klasse in Ihr System geladen, aber nur die Instanzen von LinkedList haben die Erlaubnis, diese zu verwenden.

Es zählt zu den größten Stärken von verborgenen Klassen, daß auch bei deren Verwendung für die Einführung umfassender Komplexität in die Implementierung einiger public-Klassen diese gesamte Komplexität verborgen ist, sobald die Klasse importiert und verwendet wird. Deshalb gehört zur Erstellung eines guten Pakets auch die Definition eines kleinen, sauberen Satzes von public-Klassen und Methoden, die von anderen Klassen verwendet werden können. Diese sollten dann durch einige verborgene Hilfsklassen implementiert werden.

Was sind Schnittstellen?

Schnittstellen enthalten, ebenso wie die gestern erläuterten abstrakten Klassen und Methoden, Vorlagen für Verhalten, das andere Klassen implementieren sollen. Schnittstellen bieten jedoch ein bei weitem größeres Spektrum an Funktionalität für Java und für das Klassen- und Objektdesign als einfache abstrakte Klassen und Methoden. Der Rest dieser Lektion erforscht die Schnittstellen: Was sind sie, warum sind sie für eine effektive Nutzung der Sprache Java wichtig und wie lassen sie sich implementieren und verwenden?

Das Problem der Einfachvererbung

Wenn man erstmals mit dem Design objektorientierter Programme beginnt, erscheint einem die Klassenhierarchie fast wie ein Wunder. Innerhalb dieses einzelnen Baumes können viele verschiedene Elemente ausgedrückt werden. Nach längeren Überlegungen und größerer praktischer Design-Erfahrung entdecken Sie jedoch vermutlich, daß die reine Simplizität der Klassenhierarchie einschränkend ist, insbesondere wenn Sie einige Verhalten verwenden, die von den Klassen in verschiedenen Verzweigungen derselben Struktur verwendet werden.

Lassen Sie uns einige Beispiele betrachten, die diese Probleme verdeutlichen. Am 2. Tag, als Sie die Klassenhierarchie erstmals kennengelernt haben, wurde die Vehicle- Hierarchie erläutert, siehe Abb. 15.1.

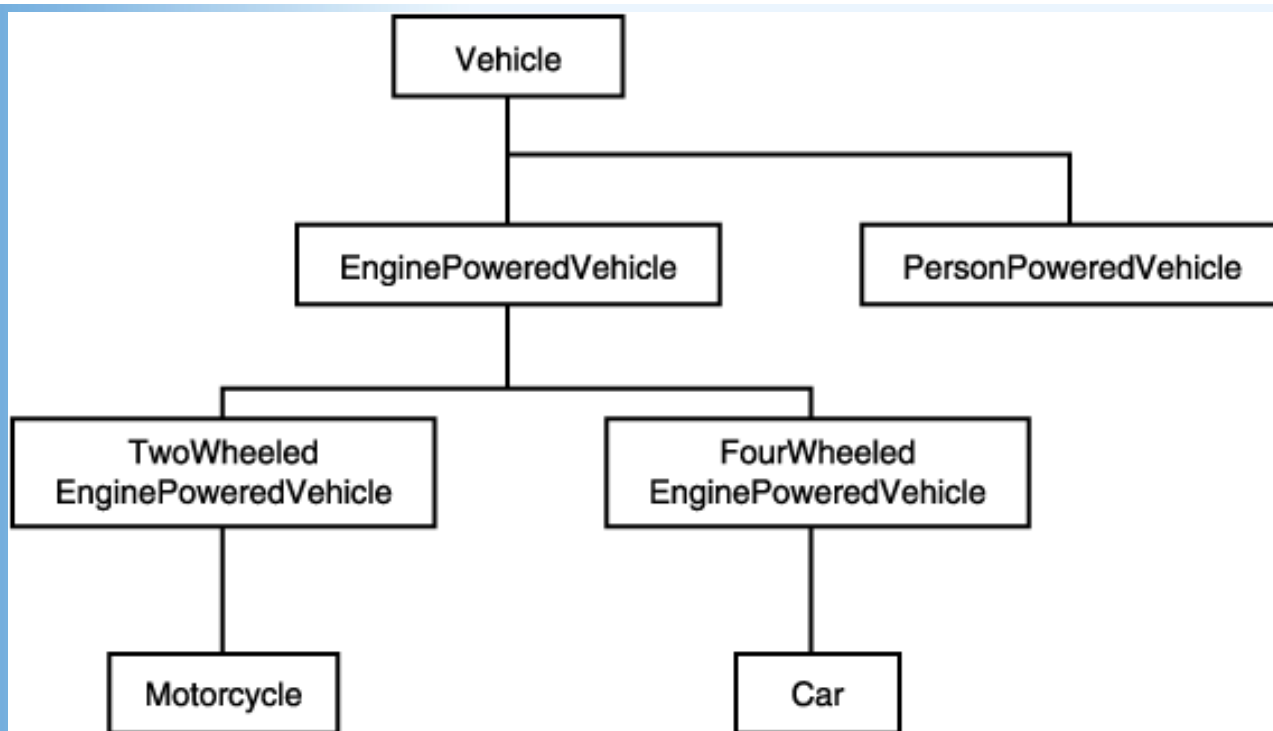


Abbildung 15.1:
Die Vehicle-Hierarchie

Dieser Hierarchie sollen nun die Klassen **BritishCars** und **BritishMotorcycle** jeweils unterhalb von **Car** und unterhalb von **Motorcycle** hinzugefügt werden. Das Verhalten, das ein Auto oder ein Motorrad britisch macht (das eventuell Methoden für `leakOil()` oder `electricalSystemFailure()` enthält), ist diesen beiden Klassen gemeinsam, aber da sie in verschiedenen Bereichen der Klassenhierarchie angesiedelt sind, läßt sich für beide keine gemeinsame Superklasse erstellen. Sie können das **British**-Verhalten in der Hierarchie auch nicht heraufsetzen, weil dieses Verhalten allen Motorrädern und Autos gemeinsam ist. Wenn Sie das Verhalten zwischen diesen beiden Klassen nicht physikalisch kopieren möchten (und damit die Regeln der objektorientierten Programmierung [OOP] für die Wiederverwendung von Codes und gemeinsames Verhalten brechen), wie können Sie dann eine solche Hierarchie erstellen?

Lassen Sie uns einen Blick auf ein schwierigeres Beispiel werfen. Angenommen Sie haben eine biologische Hierarchie mit Tieren am Anfang erstellt und darunter befinden sich die Klassen **Säugetiere** und **Vögel**. Zu den Merkmalen, die ein Säugetier definieren, gehören das Gebären von lebenden Jungen und ein Fell. Das wesentliche Kennzeichen von Vögeln ist, daß Sie einen Schnabel haben und Eier legen. Soweit, so gut. Wie können Sie nun eine Klasse für ein Schnabeltier erstellen, das sowohl Fell als auch Schnabel hat und Eier legt? Sie müßten das Verhalten von zwei Klassen kombinieren, um die Schnabeltier-Klasse zu erstellen. Da Klassen in Java aber nur eine unmittelbare Superklasse haben können, läßt sich diese Art von Problemen nicht elegant lösen.

Andere OOP-Sprachen enthalten eine breiter gefächerte Vererbung, mit der sich solche Probleme lösen lassen. Bei mehrfacher Vererbung kann eine Klasse von mehr als einer Superklasse erben und das Verhalten und die Attribute von allen seinen Superklassen gleichzeitig übernehmen. Bei mehrfacher Vererbung könnten Sie das gemeinsame Verhalten von **BritishCar** und **BritishMotorcycle** in einer einzigen Klasse (**BritishThing**) zusammenfassen und dann neue Klassen erstellen, die sowohl von der primären Superklasse als auch von der **BritishThing**-Klasse erben.

Das Problem der Mehrfachvererbung besteht darin, daß eine Programmiersprache dadurch äußerst komplex wird, dies betrifft das Lernen, die Verwendung und die Implementierung. Die Fragen zum Aufruf von Methoden und zur Organisation der Klassenhierarchie werden bei einer Mehrfachvererbung deutlich komplizierter. Zweideutigkeiten und Verwirrungen sind dann Tür und Tor geöffnet. Deshalb beschloß man, dieses Element zugunsten einer Einfachvererbung auszuschließen.

Wie läßt sich also das Problem von allgemeinem Verhalten lösen, das nicht in den strengen Rahmen der Klassenhierarchie paßt? Java, in Anlehnung an Objective-C, verwendet eine weitere Hierarchie, die aber von der Hauptklassenhierarchie verschieden ist - eine Hierarchie für gemischtes Klassenverhalten. Wenn Sie dann eine neue Klasse erstellen, verfügt diese zwar über nur eine direkte Superklasse, kann aber verschiedenes Verhalten aus anderen Hierarchien übernehmen.

Diese andere Hierarchie ist die Schnittstellenhierarchie. Eine Java-Schnittstelle ist eine Sammlung von abstraktem Verhalten, das sich in jeder beliebigen Klasse mischen läßt, um jenes Klassenverhalten hinzuzufügen, das von deren Superklassen nicht unterstützt wird. Genau genommen enthält eine Java-Schnittstelle nichts anderes als abstrakte

Methodendeklarationen und Konstanten - keine Instanzvariablen und keine Methodenimplementierungen.

Schnittstellen werden in der Klassenbibliothek von Java implementiert und verwendet, wann immer ein Verhalten wahrscheinlich von einigen anderen Klassen implementiert werden soll. Die Java-Klassenhierarchie definiert und verwendet z.B. die Schnittstellen `java.lang.Runnable`, `java.util.Enumeraion`, `java.util.Observable`, `java.awt.image.ImageConsumer` und `java.awt.imageProducer`. Einige dieser Schnittstellen haben Sie bereits kennengelernt, andere werden Sie später in diesem Buch noch entdecken. Und wieder andere sind für Ihre Programme eventuell sinnvoll, weshalb Sie in der API nachschlagen sollten, was hier für Sie zur Verfügung steht.

Schnittstellen und Klassen

Klassen und Schnittstellen haben - trotz ihrer unterschiedlichen Definition - viele Gemeinsamkeiten. Schnittstellen werden ebenso wie Klassen in Quelldateien deklariert, eine Schnittstelle in einer Datei. Ebenso wie Klassen können Sie auch mit dem Java- Compiler in `.class`-Dateien kompiliert werden. Und in den meisten Fällen können Sie anstelle von Klassen auch eine Schnittstelle verwenden.

In beinahe allen Beispielen aus diesem Buch werden Klassennamen verwendet, die sich durch einen Schnittstellen-Namen ersetzen lassen. Java-Programmierer sprechen sogar häufig von »Klassen«, wenn sie eigentlich »Klassen oder Schnittstellen« meinen. Schnittstellen ergänzen das Leistungsvermögen von Klassen und bauen dieses weiter aus. Beide lassen sich beinahe auf dieselbe Weise behandeln. Einer der wenigen Unterschiede besteht allerdings darin, daß eine Schnittstelle nicht als Instanz verwendet werden kann: `new` kann nur eine Instanz für eine Klasse erstellen.

Schnittstellen implementieren und verwenden

Sie wissen nun, was Schnittstellen sind und warum sie so leistungsstark sind. Im folgenden soll der Blick auf die einzelnen Kodierungen geworfen werden. Schnittstellen lassen sich im wesentlichen auf zwei Arten verwenden: Sie können diese in Ihren eigenen Klassen benutzen oder eigene Schnittstellen definieren. Zunächst soll die erste Variante erläutert werden.

Das Schlüsselwort `implements`

Um eine Schnittstelle zu verwenden, fügen Sie das Schlüsselwort `implements` als Teil der Klassendefinition ein. Sie haben dies bereits bei den Threads durchgeführt und die Schnittstelle `Runnable` in Ihre Applet-Definition eingefügt:

```
// java.applet.Applet ist die Superklasse
public class Neko extends java.applet.Applet
    implements Runnable { // zusätzlich verfügt die Klasse über das Runnable-
    Verhalten
    ...
}
```

Da Schnittstellen nichts anderes als abstrakte Methoden-Deklarationen enthalten, müssen Sie diese Methoden dann in Ihre eigenen Klassen implementieren, indem Sie dieselben Methodensignaturen der Schnittstelle verwenden. Beachten Sie, daß für eine einmal eingefügte Schnittstelle alle darin enthaltenen Methoden implementiert werden müssen - Sie können nicht nur jene Methoden auswählen, die Sie benötigen. Indem Sie eine Schnittstelle implementieren, teilen Sie den Benutzern Ihrer Klasse mit, daß Sie die gesamte Schnittstelle unterstützen (auch dies ist ein Unterschied zwischen Schnittstellen und abstrakten Klassen).

Nachdem Ihre Klasse eine Schnittstelle implementiert hat, können die Subklassen dieser Klasse diese neuen Methoden erben (und diese überschreiben oder überladen), ebenso als wären diese in der Superklasse definiert. Wenn Ihre Klasse von einer Superklasse erbt, die eine bestimmte Schnittstelle implementiert, müssen Sie das Schlüsselwort `implements` nicht in die eigene Klassendefinition einfügen.

Lassen Sie uns ein einfaches Beispiel verwenden und die neue Klasse `Orange` erstellen. Angenommen, Sie haben bereits die Klasse `Fruit` und eine Schnittstelle `Fruitlike` erstellt, die darstellt, was Fruits im allgemeinen durchführen können soll. Sie möchten zum einen, daß `Orange` eine `Fruit` ist, aber es soll auch ein kugelförmiges Objekt sein, daß sich drehen und wenden läßt. Im folgenden sehen Sie, wie sich dies alles ausdrücken läßt (beachten Sie die Definitionen für diese Schnittstellen im Augenblick nicht; Sie erfahren später mehr darüber):

```
interface Fruitlike {
    void decay();
    void squish();
}
```

```

    . . .
}

class Fruit implements Fruitlike {
    private Color    myColor;
    private int      daysTilIRot;
    . . .
}

interface Spherelike {
    void  toss();
    void  rotate();
    . . .
}

class Orange extends Fruit implements Spherelike {
    . . . // toss() könnte squish() aufrufen
}

```

Beachten Sie, daß die Klasse Orange nicht mit den Worten `implements Fruitlike` versehen sein muß, weil `Fruit` bereits darüber verfügt. Es gehört zu den vorteilhaften Errungenschaften dieser Struktur, daß Sie Ihre Ansicht darüber, wovon die Klasse Orange abgeleitet werden soll (wenn z.B. plötzlich eine großartige `Sphere`-Klasse eingeführt wird) jederzeit ändern können. Dennoch wird die Klasse Orange dieselben beiden Schnittstellen verstehen:

```

private float  radius;
    . . .
}

class Orange extends Sphere implements Fruitlike {
    . . . // Die Benutzer von Orange müssen von dieser Veränderung nichts
// wissen!
}

```

Mehrere Schnittstellen implementieren

Im Gegensatz zur Einfachvererbung in der Klassenhierarchie können Sie beliebig viele Schnittstellen in Ihre eigenen Klassen einfügen. Die Klassen implementieren das kombinierte Verhalten aus allen einbezogenen Schnittstellen. Um mehrere Schnittstellen in eine Klasse einzufügen, trennen Sie deren Namen durch Kommas:

```

public class Neko extends java.applet.Applet
    implements Runnable, Eatable, Sortable, Observable {
    . . .
}

```

Beachten Sie, daß sich aus der Implementierung mehrerer Schnittstellen Komplikationen ergeben können, wenn zwei verschiedene Schnittstellen jeweils dieselbe Methode definieren. Es gibt drei Möglichkeiten, dies zu lösen:

- Wenn die Methoden in jeder Schnittstelle identische Signaturen haben, implementieren Sie eine Methode in Ihrer Klasse. Diese Methode genügt für beide Schnittstellen.
- Wenn die Methoden über verschiedene Parameterlisten verfügen, ist es ein einfacher Fall von Methodenüberladung; Sie implementieren beide Methodensignaturen und jede Definition bedient die zugehörige Schnittstellendefinition.
- Wenn die Methoden dieselbe Parameterliste haben, aber sich deren Rückgabetyt unterscheidet, können Sie keine Methode erstellen, die beiden Anforderungen genügt (das Überladen von Methoden wird durch die Parameterliste und nicht durch den Rückgabetyt gekennzeichnet). In diesem Fall würde der Versuch, eine Klasse zu kompilieren, die beide Schnittstellen implementiert, einen Compiler-Fehler erzeugen. Wenn Sie dieses Problem übergehen, kann es sein, daß Ihre Schnittstelle einige Designfehler enthält und später erneut geprüft werden muß.

Andere Verwendungen für Schnittstellen

Vergegenwärtigen Sie sich, daß Sie beinahe überall anstelle einer Klasse auch eine Schnittstelle verwenden können. Sie können also eine Variable als Schnittstellentyp deklarieren:

```
Runnable aRunnableObject = new MyAnimationClass()
```

Wenn eine Variable als Schnittstellentyp deklariert ist, bedeutet dies, daß von jedem Objekt, auf welches sich die Variable bezieht, angenommen wird, es habe diese Schnittstelle implementiert - d.h. es wird also davon ausgegangen, daß es alle Methoden versteht, die von der Schnittstelle angegeben sind. Es wird vorausgesetzt, daß das Versprechen zwischen dem Designer der Schnittstelle und dessen potentiellen Implementatoren gehalten worden ist. In diesem Fall wird also davon ausgegangen, daß Sie `aRunnableObject.run()` aufrufen können, weil `aRunnableObject` ein Objekt des Typs `Runnable` enthält.

Es ist wichtig zu wissen, daß obwohl von `aRunnableObject` eine `run()`-Methode erwartet wird, der Code bereits geschrieben werden kann, lange bevor Klassen erstellt und implementiert werden. In der traditionellen, objektorientierten Programmierung ist man gezwungen, eine Klasse mit »Stub«-Implementierungen (leere Methoden oder Methoden, die sinnlose Meldungen ausgeben) zu erstellen, um die gleiche Wirkung zu erzielen.

Sie können Objekte auch für eine Schnittstelle bereitstellen, ebenso wie Sie Objekte für Klassen bereitstellen können. Lassen Sie uns für dieses Beispiel zur Definition der `Orange`-Klasse zurückkehren, die sowohl die `Fruitlike`-Schnittstelle (durch deren Superklasse `Fruit`) als auch die `Spherelike`-Schnittstelle implementiert. Hier werden die Instanzen von `Orange` für beide Klassen und Schnittstellen definiert:

```
Orange      anOrange      = new Orange();
Fruit       aFruit        = (Fruit)anOrange;
Fruitlike   aFruitlike    = (Fruitlike)anOrange;
Spherelike  aSpherelike   = (Spherelike)anOrange;
aFruit.decay();           // fruits decay()
aFruitlike.squish();       // und squish()

aFruitlike.toss();         // Dinge, die Fruitlike implementieren,
                           // implementieren toss()nicht;
aSpherelike.toss()        // die Dinge, die Spherelike implementieren,
                           // implementieren toss()
anOrange.decay();         // oranges können das alles
anOrange.squish();
anOrange.toss();
anOrange.rotate();
```

In diesem Beispiel wird eine `Orange` durch Deklarationen auf die Fähigkeiten einer Frucht oder Kugel eingeschränkt.

Beachten Sie schließlich, daß sich Schnittstellen zwar im allgemeinen mit dem Verhalten anderer Klassen (Methodensignaturen) mischen lassen, sich aber auch mit allgemein sinnvollen Konstanten mischen lassen. Wenn also zum Beispiel eine Schnittstelle als Satz von Konstanten definiert ist, und mehrere Klassen dann diese Konstanten verwendet haben, könnten die Werte dieser Konstanten global geändert werden, ohne viele Klassen einzeln ändern zu müssen. Dies ist auch ein weiteres Beispiel dafür, wo sich durch die Verwendung von Schnittstellen zur Trennung zwischen Design und Implementierung ein Code verallgemeinern und einfacher gestalten läßt.

Schnittstellen definieren und ableiten

Wenn Sie einige Zeit mit Schnittstellen gearbeitet haben, besteht der nächste Schritt darin, eigene Schnittstellen zu definieren. Schnittstellen sind den Klassen sehr ähnlich und sie werden beinahe in derselben Weise deklariert und in einer Hierarchie angeordnet, aber für die Deklaration von Schnittstellen gibt es Regeln, die befolgt werden müssen.

Neue Schnittstellen

Um eine neue Schnittstelle zu erstellen, deklarieren Sie folgendes:

```
public interface Growable {
    ...
}
```

Dies ist im Grunde dasselbe wie eine Klassendefinition, wobei das Wort `interface` das Wort `class` ersetzt. Innerhalb der Schnittstellendefinition befinden sich die Methoden und Konstanten. Die Methodendefinitionen innerhalb einer Schnittstelle sind `public`- und `abstract`-Methoden. Sie können diese explizit als solche deklarieren oder sie werden in `public`- und `abstract`-Methoden verwandelt, wenn Sie diese Modifier nicht einfügen. Eine Methode innerhalb einer Schnittstelle läßt sich nicht als `private` oder `protected` deklarieren. Im folgenden Beispiel ist die `Growable`-Schnittstelle sowohl `public` als auch `abstract` (`growIt()`) und eine ist implizit als solche deklariert (`growItBigger()`).

```
public interface Growable {
    public abstract void growIt(); //explizit public und abstract
    void growItBigger();          // effektiv public und abstract
}
```

Beachten Sie, daß ebenso wie bei abstrakten Methoden in Klassen, auch die Methoden innerhalb von Schnittstellen keinen Rumpf haben. Eine Schnittstelle ist Design in Reinform; es gibt keine Implementierungen.

Neben den Methoden können Schnittstellen auch Variablen enthalten, aber diese Variablen müssen `public`, `static` und `final` deklariert sein. Ebenso wie bei Methoden können Sie eine Variable explizit als `public`, `static` und `final` deklarieren oder diese implizit als solche definieren, wenn keiner diese Modifier verwendet wird. Im folgenden finden Sie dieselbe `Growable`-Definition mit zwei neuen Variablen:

```
public interface Growable {
    public static final int increment = 10;
    long maxnum = 1000000; // wird public static und final

    public abstract void growIt(); //explizit public und abstract
    void growItBigger();          // effektiv public und abstract
}
```

Schnittstellen müssen entweder als `public` oder ohne Modifier deklariert sein. Beachten Sie jedoch, daß Schnittstellen ohne `public`-Modifier ihre Methoden nicht automatisch in `public` und `abstract` konvertieren und auch deren Konstanten nicht in `public` konvertiert werden. Eine nichtöffentliche Schnittstelle verfügt auch über nichtöffentliche Methoden und Konstanten, die sich nur von Klassen und anderen Schnittstellen desselben Pakets verwenden lassen.

Schnittstellen können ähnlich wie Klassen zu einem Paket gehören, wenn in der ersten Zeile der Klassendatei die `package`-Anweisung eingefügt wird. Schnittstellen können auch andere Schnittstellen und Klassen aus anderen Paketen importieren, ebenso wie dies bei Klassen möglich ist.

Methoden innerhalb von Schnittstellen

Zu Methoden innerhalb von Schnittstellen ist folgender Trick anzumerken: Diese Methoden sollten abstrakt sein und einer beliebigen Klasse zugeordnet werden können, aber wie lassen sich die Parameter für diese Methoden definieren? Sie wissen ja nicht, welche Klasse sie verwendet!

Die Antwort liegt in der Tatsache, daß Sie einen Schnittstellennamen überall dort verwenden können, wo Sie einen Klassennamen benutzen - wie Sie bereits gelernt haben. Indem sie Ihre Methodenparameter als Schnittstellentypen definieren, erzeugen Sie generische Parameter, die sich allen Klassen zuweisen lassen, die diese Schnittstelle eventuell verwenden.

Als Beispiel dient die Schnittstelle `Fruitlike`, die Methoden (ohne Argumente) für `decay()` und `squish()` definiert. Hier könnte es auch die Methode für `germinateSeeds()` geben, die ein Argument hat: die Frucht selbst. Welchem Typus sollte dieses Argument angehören? Es kann nicht einfach `Fruit` sein, weil es eine Klasse wie `Fruitlike` (welche die Schnittstelle `Fruitlike` implementiert) geben könnte, die aber kein `Fruit`-Objekt ist. Die Lösung besteht darin, einfach das Argument als `Fruitlike` in der Schnittstelle zu deklarieren:

```
public interface Fruitlike {
    public abstract germinate(Fruitlike self) {
        ...
    }
}
```

In der tatsächlichen Implementierung für diese Methode in einer Klasse können Sie das generische Argument `Fruitlike` aufgreifen und an das geeignete Objekt weiterreichen:

```
public class Orange extends Fruit {
```



```

    public germinate(Fruitlike self) {
        Orange theOrange = (Orange)self;
        ...
    }
}

```

Schnittstellen ableiten

Schnittstellen sind ebenso wie Klassen in einer Hierarchie organisiert. Wenn eine Schnittstelle von einer anderen Schnittstelle erbt, übernimmt diese Subschnittstelle alle Methodendefinitionen und Konstanten der »Superschnittstelle«. Um eine Schnittstelle abzuleiten, verwenden Sie das Schlüsselwort `extends` ebenso wie in einer Klassendefinition:

```

public interface Fruitlike extends Foodlike {
    ...
}

```

Beachten Sie, daß die Schnittstellenhierarchie im Gegensatz zur Klassenhierarchie kein Äquivalent für die `Object`-Klasse besitzt; diese Hierarchie verzweigt sich nicht bis zu irgendeinem Punkt. Schnittstellen können entweder ganz selbständig bestehen oder von anderen Schnittstellen abgeleitet sein.

Ein weiterer Unterschied zur Klassenhierarchie besteht darin, daß die Vererbungshierarchie eine mehrfache Vererbung beinhaltet. Eine einfache Schnittstelle kann sich also auf die benötigte Anzahl von Schnittstellen ableiten (durch Kommas im `extends`- Teil der Definition getrennt), und die neue Schnittstelle enthält eine Kombination aller übergeordneten Methoden und Konstanten. Im folgenden finden Sie eine Schnittstellendefinition für eine Schnittstelle namens `BusyInterface`, die von allen anderen Schnittstellen erbt:

```

public interface BusyInterface extends Runnable, Growable, Fruitlike, Observable {
    ...}

```

Bei mehrfach vererbenden Schnittstellen gelten dieselben Regeln für Namenskonflikte wie bei Klassen, die mehrere Schnittstellen verwenden. Methoden, bei denen sich lediglich der Rückgabebetyp unterscheidet, erzeugen einen Compiler-Fehler.

Ein Beispiel: Verkettete Listen

Um die heutige Lektion abzuschließen, finden Sie im folgenden ein Beispiel, das Pakete und Paketschutz verwendet und eine Klasse definiert, die die Enumeration-Schnittstelle implementiert (Teil des `java.util`-Pakets). Listing 15.3 zeigt den Code.

Listing 15.3: Der gesamte Quelltext von `LinkedList.java`

```

1: package collections;
2:
3: public class LinkedList {
4:     private Node root;
5:
6:     // ...
7:     public Enumeration enumerate() {
8:         return new LinkedListEnumerator(root);
9:     }
10: }
11:
12: class Node {
13:     private Object contents;
14:     private Node next;
15:
16:     // ...
17:     public Object contents() {
18:         return contents;
19:     }
20:

```



```

21:     public Node next() {
22:         return next;
23:     }
24: }
25:
26: class LinkedListEnumerator implements Enumeration {
27:     private Node currentNode;
28:
29:     LinkedListEnumerator(Node root) {
30:         currentNode = root;
31:     }
32:
33:     public boolean hasMoreElements() {
34:         return currentNode != null;
35:     }
36:
37:     public Object nextElement() {
38:         Object anObject = currentNode.contents();
39:
40:         currentNode = currentNode.next();
41:         return anObject;
42:     }
43: }

```

Im folgenden finden Sie eine typische Verwendung für die Aufzählung:

```

collections.LinkedList aLinkedList = createLinkedList();
java.util.Enumeration e = aLinkedList.enumerate();

while (e.hasMoreElements()) {
    Object anObject = e.nextElement();
    // etwas sinnvolles mit anObject anstellen
}

```

Beachten Sie, daß wir Enumeration e zwar so benutzen, als wüßten wir, was es ist - wir wissen es aber nicht. Es handelt sich um eine Instanz einer verborgenen Klasse (LinkedListEnumeration), die man nicht direkt sehen oder benutzen kann. Durch eine Kombination von Paketen und Schnittstellen gelingt es der LinkedList-Klasse, eine transparente public-Schnittstelle für eines ihrer wichtigsten Verhalten (über die bereits definierte Schnittstelle java.util.Enumeration) bereitzustellen, während ihre zwei Implementierungsklassen nach wie vor gekapselt (verborgen) sind.

Die Weitergabe eines Objekts auf diese Art nennt man Vending. Meist gibt der »Vendor« ein Objekt weiter, das der Empfänger nicht selbst erstellen kann, aber weiß, wie es zu benutzen ist. Durch Zurückgeben des Objekts an den Vendor kann der Empfänger beweisen, daß er gewisse Fähigkeiten hat und verschiedene Aufgaben ausführen kann - und das alles, ohne viel über das weitergegebene Objekt zu wissen. Das ist ein leistungsstarkes Konzept, das in vielen Situationen anzuwenden ist.

Interne Klassen

Die meisten Java-Klassen wurden auf der Paketebene definiert, das bedeutet, daß jede Klasse ein Mitglied eines speziellen Pakets ist. Selbst wenn Sie keine explizite Verbindung zwischen einem Paket und einer Klasse herstellen, wird das Standardpaket vorausgesetzt. Klassen, die auf der Paketebene definiert sind, werden Top-Level-Klassen genannt.

Vor Java 1.1 waren Top-Level-Klassen die einzigen Klassentypen, die unterstützt wurden. Doch Java 1.1 hat einen offeneren Zugang zu den Klassendefinitionen. Java 1.1 unterstützt interne Klassen; dies sind Klassen, die sich für jeden beliebigen Zweck definieren lassen. Das heißt, eine Klasse kann als Mitglied einer anderen Klasse definiert werden oder innerhalb eines Anweisungsblocks bzw. anonym in einem Ausdruck.

Listing 15.4 beinhaltet ein Applet, das den Namen Inner, das eine interne Klasse namens BlueButton verwendet. Diese Klasse repräsentiert Schaltflächen, deren Hintergrundfarbe standardmäßig blau ist.

Listing 15.4: Der gesamte Quelltext von Inner.java

```

1: import java.awt.Button;
2: import java.awt.Color;
3:
4: public class Inner extends java.applet.Applet {
5:     Button b1 = new Button("One");
6:     BlueButton b2 = new BlueButton("Two");
7:
8:     public void init() {
9:         add(b1);
10:        add(b2);
11:    }
12:    class BlueButton extends Button {
13:        BlueButton(String label) {
14:            super(label);
15:            this.setBackground(Color.blue);
16:        }
17:    }
18: }

```

Die Abbildung 15.2 wurde mit dem Appletviewer erzeugt. Das Applet wurde über den folgenden HTML-Code in die Webseite integriert:

```

<applet code="Inner.class" width=100 height=100>
</applet>

```



Abbildung 15.2:
Das Inner-Applet

In diesem Beispiel unterscheidet sich die Klasse BlueButton nicht von einer Hilfsklasse, die sich in derselben Quelldatei befindet, in der sich auch die Hauptklasse des Programms befindet. Der einzige Unterschied ist, daß die Hilfsklasse in der Hauptklasse selbst definiert ist, was einige Vorteile hat:

- Interne Klassen sind für alle anderen Klassen nicht sichtbar. Das heißt Sie müssen sich keine Gedanken um Namenskonflikte zwischen dieser und anderen Klassen machen.
- Interne Klassen können auf Methoden und Variablen im Gültigkeitsbereich der top-level-Klasse zugreifen, auf die sie als eigenständige Klasse nicht zugreifen könnten.

In vielen Fällen ist eine interne Klasse eine kleine Klasse, die nur für eine sehr eingeschränkte Aufgabe zuständig ist. In dem Inner-Applet ist die Klasse BlueButton gut für die Implementierung als interne Klasse geeignet, da sie kaum komplexe Verhaltensweisen und Attribute enthält.

Der Name der internen Klasse ist mit dem Namen der Klasse verbunden, die die interne Klasse beinhaltet. Dieser wird bei der Kompilierung des Programms automatisch zugewiesen. Im BlueButton-Beispiel wird der Name Inner\$BlueButton.class vom JDK vergeben.



Wenn Sie interne Klassen verwenden, müssen Sie noch stärker darauf achten, daß Sie alle .class-Dateien mitliefern, wenn Sie ein Programm verfügbar machen. Jede interne Klasse hat ihre eigene .class-Datei und diese müssen zusammen mit der

jeweiligen top-level-Klasse zur Verfügung gestellt werden. Wenn Sie z.B. das Inner-Applet veröffentlichen würden, dann müßten Sie sowohl die Datei Inner.class als auch die Datei Inner\$BlueButton.class veröffentlichen.

Interne Klassen scheinen nur eine kleine Erweiterung der Sprache Java zu sein. Tatsächlich stellen Sie eine wesentliche Änderung der Sprache dar.

Regeln, die den Gültigkeitsbereich einer internen Klasse betreffen, decken sich fast mit denen der Variablen. Der Name einer internen Klasse ist außerhalb ihres Gültigkeitsbereichs nicht sichtbar - außer in einer vollständigen Namensangabe. Dies hilft bei der Strukturierung von Klassen in einem Paket. Der Code einer internen Klasse kann einfach die Namen der umgebenden Gültigkeitsbereiche verwenden. Darunter fallen sowohl Klassen und Variablen von umgebenden Klassen als auch lokale Variablen umgebender Blocks.

Zusätzlich können Sie eine Top-Level-Klasse als ein static-Mitglied einer anderen Top-Level-Klasse definieren. Anders als eine interne Klasse kann eine Top-Level-Klasse nicht direkt die Instanz-Variablen einer anderen Klasse verwenden. Die Möglichkeit, Klassen auf diese Art ineinander zu verschachteln, erlaubt es einer Top-Level-Klasse, eine Art Paketorganisation für logisch zueinander in Bezug stehende Top-Level-Klassen der zweiten Reihe zu bieten.

Zusammenfassung

Heute haben Sie gelernt, wie Sie ein Objekt mit Hilfe von Modifiern für die Zugriffskontrolle auf dessen Methoden und Variablen kapseln. Sie haben auch gelernt, wie Sie die Modifier static, final und abstract bei der Entwicklung von Java-Klassen und Klassenhierarchien verwenden.

Um den Aufwand der Entwicklung und Verwendung einer Reihe von Klassen zu unterstützen, haben Sie gelernt, wie Klassen in Paketen gruppiert werden können. Diese Gruppen helfen Ihnen dabei, Ihre Programme besser zu organisieren, und erlauben es Ihnen, Ihre Klassen mit den vielen Java-Programmierern zu teilen, die ihren Code öffentlich verfügbar machen.

Schließlich haben Sie noch gelernt, wie Sie Schnittstellen und interne Klassen implementieren. Dabei handelt es sich um zwei Strukturen, die beim Entwurf einer Klassenhierarchie hilfreich sind.

Fragen und Antworten

Frage:

Ich fürchte, daß die intensive Verwendung von Accessor-Methoden meinen Java- Code verlangsamt. Stimmt das?

Antwort:

Nicht unbedingt. Demnächst sind Java-Compiler klug genug, um alles automatisch zu beschleunigen. Wenn Sie sich aber über die Geschwindigkeit Sorgen machen, können Sie Accessor-Methoden final deklarieren, dann laufen sie so schnell wie direkte Instanzvariablen.

Frage:

Unterliegen static-Methoden der gleichen Vererbung wie Instanzmethoden?

Antwort:

Nein. Klassenmethoden sind standardmäßig final. Das bedeutet, daß Sie keine Klassenmethode als nicht final deklarieren können! Die Vererbung von Klassenmethoden ist nicht zulässig, was die Symmetrie zu Instanzmethoden bricht.

Frage:

Sofern ich die letzte Lektion richtig verstanden habe, scheinen final-abstract- oder private-abstract-Methoden oder -Klassen unsinnig zu sein. Sind sie überhaupt zulässig?

Antwort:

Nein, sind sie nicht. Das haben Sie richtig erkannt; sie führen zu Kompilierfehlern. Um überhaupt brauchbar zu sein, müssen abstract-Methoden überschrieben und von abstract-Klassen müssen Subklassen angelegt werden. Beide Operationen sind aber unzulässig, falls sie gleichzeitig auch public oder final sind.

Woche 3

Tag 16

Ausnahmezustände: Fehlerbehandlung und Sicherheit

Programmierer aller Sprachen bemühen sich, fehlerfreie Programme zu schreiben, Programme, die nie abstürzen, Programme, die jede Situation mit Eleganz behandeln können und ungewöhnliche Situationen in den Griff bekommen, ohne daß der Benutzer eingreifen müßte. Gute Vorsätze hin und her - solche Programme gibt es nicht.

In realen Programmen treten Fehler auf, weil entweder der Programmierer nicht an jede Situation gedacht hat, in die das Programm kommen kann (oder er hatte nicht die Zeit, das Programm ausgiebig genug zu testen), oder weil Situationen auftreten, die sich der Kontrolle des Programmierers entziehen - schlechte Daten vom Benutzer, beschädigte Dateien, die nicht die richtigen Daten beinhalten, Geräte, die nicht antworten, Sonnenflecken, Gremlins, was auch immer.

In Java wird diese Art seltsamer Ereignisse, die Fehler in einem Programm auslösen können, als Ausnahmen (engl. Exceptions) bezeichnet. Java bietet in der Sprache einige Features, die sich mit den Ausnahmen beschäftigen. Darunter sind die folgenden:

- So behandeln Sie Ausnahmen in Ihrem Code und beheben mögliche Probleme elegant
- So teilen Sie Java und den Benutzern Ihrer Methoden mit, daß Sie eine mögliche Ausnahme erwarten
- Wo die Einschränkungen in Ihrem Code liegen und wie Sie diesen dennoch robuster gestalten

Neben den Ausnahmen lernen Sie auch das System kennen, das mit Java 1.2 eingeführt wurde und es Applets ermöglicht, Dinge zu tun, die normalerweise Sicherheitsausnahmen erzeugen.

Programmieren im großen

Mit zunehmender Erfahrung im Java-Programmieren werden Sie feststellen, daß Sie nach dem Design der Klassen und Schnittstellen sowie der Methodendefinitionen immer noch keine Eigenschaften für Ihre Objekte definiert haben. Schließlich beschreibt eine Schnittstelle die übliche Verwendungsweise eines Objekts, beinhaltet aber keine seltsamen Ausnahmefälle. Bei vielen Systemen wird dieses Problem in der Dokumentation gelöst, z.B. durch Auflisten von Rückgabewerten, wie im vorherigen Beispiel. Da dem System darüber nichts bekannt ist, kann es keine Konsistenzprüfung durchführen. Der Compiler kann bei solchen Ausnahmebedingungen in keiner Weise eingreifen, im Gegensatz zu den hilfreichen Warnungen und Fehlermeldungen, die er produziert, wenn eine Methode fehlerhaft ist.

Dieser Aspekt wird im Programmdesign nicht berücksichtigt. Vielmehr sind Sie gezwungen, das irgendwie in der Dokumentation zu beschreiben, in der Hoffnung, daß keiner später bei der Implementierung einen Fehler macht. Das wird dadurch noch erschwert, daß jeder die gleiche Sache anders beschreibt. Sie benötigen also eine einheitliche Form der Deklaration der Absichten von Klassen und Methoden in bezug auf diese Ausnahmebedingungen. Java bietet eine solche Möglichkeit:

```
public class MyFirstExceptionalClass {  
    public void anExceptionalMethod() throws MyFirstException {  
        ...  
    }  
}
```


}

Hier wird der Leser (und der Compiler) darauf aufmerksam gemacht, daß der Code ... eine Ausnahme namens `MyFirstException` auswerfen kann.

Sie können sich die Beschreibung einer Methode als Vertrag zwischen dem Designer und der Methode (oder Klasse) und sich selbst als Aufrufer der Methode vorstellen. Normalerweise teilt diese Beschreibung die Typen der Argumente einer Methode, was sie ausgibt, und die allgemeine Semantik mit. Ihnen wird ebenfalls mitgeteilt, welche abnormen Dinge sie ausführen kann. Das ist ein Versprechen, genauso wie die Methode verspricht, einen Wert eines bestimmten Typs auszugeben, und Sie sich darauf verlassen können, wenn Sie den Code schreiben. Diese neuen Versprechen helfen, alle Stellen, an denen Ausnahmebedingungen in Ihrem Programm gehandhabt werden sollen, explizit zu bezeichnen.

Da Ausnahmen Instanzen von Klassen sind, können sie in eine Hierarchie gestellt werden, die auf natürliche Weise die Beziehungen zwischen den verschiedenen Ausnahmearten beschreibt. Wenn Sie sich die Klassenhierarchiediagramme von `java.lang`-Fehlern und `java.lang`-Ausnahmen ansehen, werden Sie feststellen, daß unter der Klasse `Throwable` zwei große Klassenhierarchien stehen. Die Wurzeln dieser zwei Hierarchien sind Subklassen von `Throwable` namens `Exception` und `Error`. Diese Hierarchien verkörpern die reichhaltigen Beziehungen, die zwischen Ausnahmen und Fehlern in der Java-Laufzeitumgebung bestehen.

Wenn Sie wissen, daß eine bestimmte Fehler- oder Ausnahmeart in Ihrer Methode auftreten kann, müssen Sie diese entweder selbst handhaben oder die potentiellen Aufrufer explizit mit der `throws`-Klausel darauf aufmerksam machen. Sie müssen nicht alle Fehler und Ausnahmen auflisten. Instanzen der Klasse `Error` oder `RuntimeException` (eine ihrer Subklassen) müssen in der `throws`-Klausel nicht aufgeführt werden. Sie werden besonders behandelt, weil sie irgendwo in einem Java-Programm auftreten können und normalerweise durch Bedingungen verursacht werden, die nicht auf den Programmierer zurückzuführen sind. Ein gutes Beispiel dafür ist `OutOfMemoryError`, ein Fehler, der jederzeit aus vielen Gründen auftreten kann.

Sie können diese Fehler und Laufzeitausnahmen auf Wunsch selbstverständlich auflisten, dann sind die Aufrufer Ihrer Methoden gezwungen, sie zu behandeln. Wenn das Wort »Exception« irgendwo allein steht, bedeutet es fast immer »Ausnahme oder Fehler« (d.h. eine `Throwable`-Instanz). In der obigen Diskussion wurde erklärt, daß Ausnahmen und Fehler im Grunde zwei getrennte Hierarchien bilden, sich abgesehen von der `throws`-Regel aber gleich verhalten.

Wenn Sie sich die Diagramme in Anhang B genauer ansehen, werden Sie feststellen, daß es nur fünf Ausnahmearten (in `java.lang`) gibt, die in einer `throws`-Klausel aufgelistet werden müssen (bedenken Sie, daß alle `Errors` und `RuntimeExceptions` Ausnahmen sind):

- `ClassNotFoundException`
- `IllegalAccessException`
- `InstantiationException`
- `InterruptedException`
- `NoSuchMethodException`

Jeder dieser Namen deutet etwas an, das explizit vom Programmierer veranlaßt wird, nicht etwas, das hinter den Kulissen abläuft, wie etwa `OutOfMemoryError`.

Unterhalb von `java.util` und `java.io` in der Klassenhierarchie sehen Sie, daß jedes Paket neue Ausnahmen hinzufügt. `java.util` fügt die zwei Ausnahmen `ArrayStoreException` und `IndexOutOfBoundsException` hinzu und stellt sie unter `RuntimeException`. `java.io` fügt einen ganzen Baum von `IOExceptions` hinzu, die eher vom Programmierer verursacht werden und deshalb unter der Wurzel `Exception` eingereiht werden. Somit muß `IOExceptions` in `throws`-Klauseln beschrieben werden. Schließlich definiert das Paket `java.awt`

jeweils einen impliziten und einen expliziten Stil.

Die Java-Klassenbibliothek nutzt Ausnahmen überall sehr wirkungsvoll. Wenn Sie sich die ausführliche API-Dokumentation Ihres Java-Releases ansehen, sehen Sie, daß viele Methoden in der Bibliothek throws-Klauseln haben. Einige davon sind sogar dokumentiert (um sie dem Leser klarer darzustellen). Das ist lediglich auf die Nettigkeit des Verfassers zurückzuführen, denn von Ihnen wird nicht erwartet, in Ihren Programmen derartige Bedingungen zu berücksichtigen.

Programmieren im kleinen

Sie haben inzwischen schon ein gutes Gefühl bekommen, auf welche Weise Ausnahmen das Design eines Programms und einer Klassenbibliothek verbessern können. Wie aber werden Ausnahmen praktisch angewandt? Wir wollen das nun mit `anotherExceptionalMethod()` aus dem ersten Beispiel der heutigen Lektion versuchen:

```
public void anotherExceptionalMethod() throws MyFirstException {
    MyFirstExceptionalClass aMFEC = new MyFirstExceptionalClass();
    aMFEC.anExceptionalMethod();
}
```

Wir betrachten dieses Beispiel jetzt genauer. Da `MyFirstException` eine Subklasse von `Exception` ist, müssen Sie sie im Code von `anotherExceptionalMethod()` verarbeiten oder andernfalls die Aufrufer entsprechend warnen. Da Ihr Code `anExceptionalMethod()` lediglich aufruft, ohne die Tatsache zu berücksichtigen, daß `MyFirstException` ausgeworfen werden könnte, müssen Sie diese Ausnahme in Ihre throws- Klausel einfügen. Das ist absolut zulässig und verschont die Aufrufer vor etwas, für das Sie eigentlich zuständig sind (was selbstverständlich von den Umständen abhängt).

Nehmen wir an, Sie fühlen sich heute verantwortlich. Sie entschließen sich, die Ausnahme zu behandeln. Da Sie jetzt eine Methode ohne throws-Klausel deklarieren, müssen Sie die erwarteten Ausnahmen mit `catch` auffangen und etwas Nützliches damit anfangen:

```
public void responsibleMethod() {
    MyFirstExceptionalClass aMFEC = new MyFirstExceptionalClass();
    try {
        aMFEC.anExceptionalMethod();
    } catch (MyFirstException m) {
        ... // Tun Sie etwas schrecklich Wichtiges
    }
}
```

Die try-Anweisung sagt praktisch alles: »Versuche, den Code innerhalb dieser Klammern auszuführen. Falls Ausnahmen ausgeworfen werden, sind entsprechende Handler dafür verfügbar.« Sie können am Ende von `try` beliebig viele `catch`-Klauseln einfügen. Mit jeder einzelnen können Sie die Ausnahmen, die in Instanzen vorkommen, handhaben. Mit `catch` in diesem Beispiel werden Ausnahmen der Klasse `MyFirstException` (oder einer ihrer Subklassen) gehandhabt.

Wenn Sie beide aufgezeigten Ansätze kombinieren, d.h. die Ausnahme selbst behandeln, sie aber auch den Aufrufern zur Kenntnis bringen wollen, können Sie das durch explizites Auswerfen der Ausnahme:

```
public void responsibleExceptionalMethod() throws MyFirstException {
    MyFirstExceptionalClass aMFEC = new MyFirstExceptionalClass();
    try {
        aMFEC.anExceptionalMethod();
    } catch (MyFirstException m) {
        ... // Etwas Verantwortungsvolles tun
    }
}
```

```
throw m; // Erneutes Auswerfen der Ausnahme
}
}
```

Das funktioniert, weil Ausnahmen-Handler darin verschachtelt sind. Sie können die Ausnahme behandeln, indem Sie etwas Verantwortungsvolles mit ihr anfangen, entschließen sich aber, keinen Ausnahmen-Handler bereitzustellen, wenn die Aufrufer selbst eine Gelegenheit haben, sie zu behandeln. Ausnahmen fließen über die gesamte Kette der Methodenaufrufe (von denen die meisten normalerweise die Ausnahmen nicht behandeln werden), bis das System zuletzt eventuell nicht aufgefangene Ausnahmen selbst behandelt, indem es das Programm beendet und eine Fehlermeldung ausgibt. In einem Einzelprogramm ist das nicht einmal so schlecht. In einem Applet aber kann das den Browser zum Absturz bringen. Die meisten Browser schützen sich vor dieser Katastrophe, indem sie alle Ausnahmen beim Ausführen eines Applets selbst auffangen, sicher ist das aber nicht. Die Faustregel lautet deshalb: Falls es Ihnen möglich ist, eine Ausnahme aufzufangen, sollten Sie das auch tun.

Wir betrachten nun, wie das Auswerfen einer neuen Ausnahme aussieht. Wir schlachten das Beispiel der ersten Lektion weiter aus:

```
public class MyFirstExceptionalClass {
public void anExceptionalMethod() throws MyFirstException {
...
if (someThingUnusualHasHappened()) {
throw new MyFirstException();
// Die Ausführung kommt nicht bis hierher
}
}
}
```

throw ist mit der break-Anweisung vergleichbar - was danach folgt, wird nicht mehr ausgeführt.

Das ist eine grundlegende Möglichkeit, alle Ausnahmen zu erzeugen. Die ganze Hierarchie unter der Klasse Throwable wäre nichts wert, wenn man die throw-Anweisung nicht überall im Code verwenden könnte. Da sich Ausnahmen über jede Tiefe bis in die Methoden hinein ausbreiten können, kann jeder Methodenaufwurf theoretisch eine Fülle möglicher Fehler und Ausnahmen erzeugen. Zum Glück muß man nur die in der throws-Klausel der jeweiligen Methode auflisten. Der Rest wandert stillschweigend zur Ausgabe einer Fehlermeldung (oder bis er vom System aufgefangen wird).

Im folgenden ungewöhnlichen Beispiel wird das aufgezeigt, wobei throw und der auffangende Handler sehr eng zusammenliegen:

```
System.out.print("Now ");
try {
System.out.print("is ");
throw new MyFirstException();
System.out.print("a ");
} catch (MyFirstException m) {
System.out.print("the ");
}
System.out.print("time.");
```

Dieser Code gibt die Zeichenkette Now is the time. aus.

Ausnahmen sind eine starke Technik, um den Bereich aller möglichen Fehlerbedingungen in handhabbare Stücke aufzuteilen. Da die erste passende catch-Klausel ausgeführt wird, können Sie Ketten wie beispielsweise folgende erstellen:

```

try {
someReallyExceptionalMethod();
} catch (NullPointerException n) { // Eine Subklasse von RuntimeException
...
} catch (RuntimeException r) { // Eine Subklasse von Exception
...
} catch (IOException i) { // Eine Subklasse von Exception
...
} catch (MyFirstException m) { // Unsere Subklasse von Exception
...
} catch (Exception e) { // Eine Subklasse von Throwable
...
} catch (Throwable t) {
... // Fehler, plus alles, was oben nicht
// aufgefangen wurde
}

```

Indem Subklassen vor ihren Elternklassen aufgelistet werden, fangen die Eltern alles auf, was nicht von einer der darüberstehenden Subklassen aufgefangen wurde. Mit Ketten dieser Art können Sie fast alle Testkombinationen ausdrücken. Findet etwas wirklich Seltsames statt, das Sie nicht behandeln können, kann dies eventuell durch Verwendung einer Schnittstelle aufgefangen werden. Dadurch können Sie Ihre Ausnahmenhierarchie unter Verwendung der Mehrfachvererbung auslegen. Das Auffangen einer Schnittstelle anstatt einer Klasse eignet sich auch zum Testen einer Eigenschaft, die in vielen Ausnahmen vorkommt, in einem Einfachvererbungsbaum aber nicht ausgedrückt werden kann.

Nehmen wir beispielsweise an, daß mehrere in Ihrem Code verteilte Ausnahmenklassen nach dem Auswerfen einen Neustart voraussetzen. Sie können eine Schnittstelle namens NeedsReboot erstellen, so daß alle Ausnahmenklassen die Schnittstelle implementieren. (Keine davon braucht eine Elternklasse.) Dann fängt der Ausnahmen- Handler auf der höchsten Ebene Klassen auf, die NeedsReboot implementieren, und führt einen Neustart aus:

```

public interface NeedsReboot { } // Braucht überhaupt keinen Inhalt
try {
someMethodThatGeneratesExceptionsThatImplementNeedsReboot();
} catch (NeedsReboot n) { // Schnittstelle auffangen
... // Aufräumen
SystemClass.reboot(); // Neustart anhand einer erfundenen
    Systemklasse
}

```

Übrigens, wenn Sie wirklich ungewöhnliche Verhaltensweisen während einer Ausnahme brauchen, können Sie diese in die Ausnahmenklasse einfügen! Denken Sie daran, daß eine Ausnahme eine normale Klasse ist. Deshalb kann sie Instanzvariablen und Methoden enthalten. Deren Verwendung ist zwar ein bißchen seltsam, kann sich bei absonderlichen Situationen aber als nützlich erweisen. Das könnte etwa so aussehen:

```

try {
someExceptionallyStrangeMethod();
} catch (ComplexException e) {
switch (e.internalState()) { // Wahrscheinlich der Wert einer
// Instanzvariablen
case e.COMPLEX_CASE: // Eine Klassenvariable der Ausnahme
e.performComplexBehavior(myState, theContext, etc);
break;
...
}

```

```
}
}
```

Einschränkungen beim Programmieren

So leistungsstark sich das alles anhört, kann man sich des Eindrucks nicht erwehren, daß es gewisse Einschränkungen auferlegt, stimmt's? Nehmen wir beispielsweise an, Sie wollen die Standardmethode `toString()` der `Object`-Klasse überschreiben, um sich einen Einblick zu verschaffen, was eigentlich alles ausgegeben wird:

```
public class MyIllegalClass {
public String toString() {
someReallyExceptionalMethod();
... // Gibt eine Zeichenkette aus
}
}
```

Da die Superklasse `Object` die Methodendeklaration für `toString()` ohne `throws`-Klausel definiert, muß eine Implementierung davon in einer Subklasse dieser Einschränkung gehorchen. Insbesondere können Sie nicht einfach `someReallyExceptionalMethod()` wie zuvor aufrufen, weil sie eine Fülle von Fehlern und Ausnahmen erzeugt, von denen einige in der `throws`-Klausel aufgelistet werden (z.B. `IOException` und `MyFirstException`). Würden keine ausgeworfenen Ausnahmen in der Liste stehen, hätten Sie kein Problem. Da einige aber darin aufgeführt werden, müssen Sie mindestens diese mit `catch` auffangen:

```
public class MyLegalClass {
public String toString() {
try {
someReallyExceptionalMethod();
} catch (IOException e) {
} catch (MyFirstException m) {
}
... // Gibt eine Zeichenkette aus
}
}
```

In beiden Fällen werden die Ausnahmen zwar aufgefangen, jedoch wird absolut nichts unternommen. Das ist zulässig, aber nicht immer die richtige Entscheidung. Man sollte sich einige Gedanken machen, um das beste nichttriviale Verhalten einer bestimmten `catch`-Klausel festzulegen. Diese zusätzliche Denkarbeit macht Ihr Programm robuster, kann ungewöhnliche Eingaben leichter handhaben und funktioniert im Zusammenhang mit Multithreading besser.

Die `toString()`-Methode von `MyIllegalClass` produziert einen Kompilierfehler. Wenn Sie sich darüber einige Gedanken machen, um die Situation optimal zu lösen, werden Sie reichlich dafür belohnt, denn Ihre Klassen können dann in späteren Projekten bzw. größeren Programmen wiederverwendet werden. Selbstverständlich wurde die Java-Bibliothek mit genau dieser Sorgfalt entwickelt. Unter anderem ist sie auch deshalb robust, so daß vielseitige Java-Projekte entwickelt werden können.

Die finally-Klausel

Nehmen wir an, es gibt eine Aktion, die Sie unbedingt ausführen müssen, egal was passiert. Normalerweise ist das das Freigeben von externen Ressourcen, die beansprucht wurden, oder eine Datei zu schließen usw. Um sicherzugehen, daß »egal was passiert« auch Ausnahmen beinhaltet, benutzen Sie eine Klausel der `try`-Anweisung, die genau für diese Zwecke entwickelt wurde - `finally`:


```

SomeFileClass f = new SomeFileClass();
if (f.open("/a/file/name/path")) {
try {
someReallyExceptionalMethod();
} finally {
f.close();
}
}

```

Diese Anwendung von finally verhält sich ungefähr so:

```

SomeFileClass f = new SomeFileClass();
if (f.open("/a/file/name/path")) {
try {
someReallyExceptionalMethod();
} catch (Throwable t) {
f.close();
throw t;
}
}

```

ausgenommen, daß finally hier auch zum Aufräumen nach Ausnahmen *und* nach return-, break- und continue-Anweisungen benutzt werden kann. Nachfolgend eine komplexe Demonstration:

```

public class MyFinalExceptionalClass extends ContextClass {
public static void main(String argv[]) {
int mysteriousState = getContext();
while (true) {
System.out.print("Who ");
try {
System.out.print("is ");
if (mysteriousState == 1)
return;
System.out.print("that ");
if (mysteriousState == 2)
break;
System.out.print("strange ");
if (mysteriousState == 3)
continue;
System.out.print("but kindly ");
if (mysteriousState == 4)
throw new UncaughtException();
System.out.print("not at all ");
} finally {
System.out.print("amusing ");
}
System.out.print("yet compelling ");
}
System.out.print("man?");
}
}

```

Dieser Code erzeugt je nach dem Wert von mysteriousState eine der folgenden Ausgaben:

- 1 Who is amusing
- 2 Who is that amusing man?
- 3 Who is that strange amusing who is that strange amusing ...
- 4 Who is that strange but kindly amusing
- 5 Who is that strange but kindly not at all amusing yet compelling man?

In Fall 3 endet die Ausgabe erst, wenn (Strg)+(C) gedrückt werden. In Fall 4 wird auch eine von `UncaughtException` erzeugte Fehlermeldung ausgegeben.

Digitale Signaturen zur Identifikation von Applets

Die Strategie für Sicherheit von Java-Applets geht davon aus, daß Sie niemandem im World Wide Web trauen können. Diese Denkweise mag zynisch klingen, in der Praxis bedeutet sie folgendes: Potentiell kann jeder ein Java-Applet erstellen, das Schaden anrichtet, der Benutzer hat keine Chance, sich dagegen zu wehren. Deshalb wurden von Anfang an alle Funktionen der Sprache, die einen Mißbrauch ermöglichen, zur Verwendung bei Applets ausgeschlossen. Darunter fallen:

- Dateien auf dem Rechner, auf dem das Applet ausgeführt wird, lesen.
- Dateien auf dem Rechner, auf dem das Applet ausgeführt wird, schreiben.
- Informationen über eine Datei auf dem System auslesen.
- Eine Datei auf dem System löschen.
- Eine Netzwerkverbindung zu einem anderen Rechner als dem, der die Webseite, auf der sich das Applet befindet, geliefert hat.
- Ein Fenster anzeigen, das keine Warnung enthält, daß es sich um ein Fenster eines Applets handelt.

Java 1.2 bietet eine Möglichkeit an, wie sich alle Funktionen, die für Java-Anwendungen zur Verfügung stehen, auch für Applets einsetzen lassen, sofern sie von einem vertrauenswürdigen Applet-Hersteller stammen und eine digitale Signatur enthalten, welche die Authentizität bestätigen. Eine *digitale Signatur* ist eine verschlüsselte Datei, die ein Programm begleitet und genau angibt, woher das Programm stammt. Benutzer, die wissen, wer ein Programm hergestellt hat, können dann selbst darüber entscheiden, ob sie dieser Firma oder dem einzelnen Hersteller vertrauen. Wer bereits mit ActiveX-Steuerelementen vertraut ist, kennt dieses System, mit dem auch ActiveX-Steuerelemente auf Seiten im World Wide Web zur Verfügung gestellt werden.

Ein Dokument, das eine digitale Signatur repräsentiert, wird Zertifikat genannt (engl. *certificate*).

Damit ein Applet-Hersteller vertrauenswürdig wird, muß er seine Identität gegenüber einer Gruppe, die als Zertifizierungsautorität (engl. *certificate authority*) bezeichnet wird, verifizieren. Idealerweise sind diese Gruppen unabhängig von den Applet-Herstellern und sollten in dem Ruf einer verlässlichen Firma stehen. Momentan bieten die folgenden Firmen Authentifizierungsdienste in irgendeiner Form an:

- VeriSign - Der erste und am weitesten verbreitete Zertifizierungsdienst. Angeboten werden sowohl Microsoft- als auch Netscape-spezifische Authentifizierungen. <http://www.verisign.com>
- Thawte Certification - Eine neue Zertifizierungsautorität für Microsoft, Netscape und Test-Zertifikate. <http://www.thawte.com>

Andere Firmen bieten Zertifizierungsdienste für Kunden in bestimmten geografischen Gebieten. Netscape listet die Zertifizierungsautoritäten, mit denen sie zusammenarbeiten, unter der folgenden Web-Adresse auf:

<https://certs.netscape.com/client.html>



Das allgemeine Sicherheitsmodell, das hier beschrieben wird, ist das offizielle, das von JavaSoft erstellt wurde. JavaSoft nutzt dieses im eigenen Browser und alle Browser, die voll kompatibel zu Java 1.2 sind, nutzen es ebenfalls. Netscape und Microsoft haben ihre eigenen Sicherheitsmodelle für die eigenen Browser vorgestellt. Das heißt, ein Applet muß für jeden Browser, in dem es laufen soll, das entsprechende System implementieren. Glücklicherweise sind sich die Systeme ähnlich, so daß es einfacher wird, die anderen zu lernen, sobald man eines gemeistert hat.

Es ist ebenfalls möglich, Sicherheitsstufen zwischen absolut vertrauenswürdig (das Applet darf alles machen) und kein Vertrauen (das Applet darf nichts tun, was eventuell Schaden anrichten könnte) festzulegen. Java 1.2 ermöglicht dies über eine Reihe von Klassen, die als Permissions bezeichnet werden.

Im Moment unterliegen Applets allen Einschränkungen, bis der Entwickler das Applet digital signiert und der Benutzer diesen Entwickler als vertrauenswürdig einstuft.

Ein Beispiel mit einer digitalen Signatur

Um das Prozedere für eine sichere Applet-Übertragung besser zu verstehen, soll für dieses Beispiel folgendes Szenario gelten: Der Entwickler des Applets heißt *Fishhead Software*, es gibt eine unabhängige Java-Gruppe namens *Signatures R`US* und einen Web-Benutzer namens *Gilbert*.

Fishhead Software bietet auf seiner Website ein Applet-Spiel an, das High Scores und andere Informationen auf die Festplatte des Benutzers schreibt. Dies ist normalerweise nicht möglich, weil die Applet-Beschränkungen einen Zugriff auf die Festplatte des Benutzers ausschließen. Damit das Spiel dennoch gespielt werden kann, muß *Fishhead Software* das Applet mit einer digitalen Signatur versehen. Auf diese Weise können die Benutzer *Fishhead Software* als vertrauenswürdigen Software-Hersteller identifizieren.

Dieser Vorgang wird in fünf Schritten ausgeführt:

1. *Fishhead Software* verwendet *keytool*, ein Tool aus dem JDK, welches zwei verschlüsselte Dateien erstellt, die *öffentlicher Schlüssel* und *privater Schlüssel* heißen. Zusammen bilden beide Schlüssel eine elektronische »ID-Karte«, mit deren Hilfe sich der Hersteller komplett identifizieren läßt. *Fishhead Software* stellt sicher, daß niemand anders auf den *privaten Schlüssel* Zugriff erhält. Der *öffentliche Schlüssel* kann und sollte jedem als Teil der ID zugänglich gemacht werden.
2. *Fishhead Software* benötigt eine Instanz, die beglaubigen kann, wer die Firma ist. Es schickt seinen *öffentlichen Schlüssel* und eine beschreibende Datei zu *Fishhead Software* an eine unabhängige Gruppe, der die Java-Benutzer trauen. Diese Gruppe ist *Signatures R`US*.
3. *Signatures ,R`US* überprüft *Fishhead Software* mit dem *öffentlichen Schlüssel*, um sicherzustellen, daß es sich um eine legitime Software-Firma handelt. Wenn *Fishhead Software* Muster verschickt, erstellt *Signatures R`US* eine neue verschlüsselte Datei, die Zertifikat genannt wird. Dieses Zertifikat wird an *Fishhead Software* zurückgeschickt.
4. *Fishhead* erstellt eine Java-Archivdatei für das Applet-Spiel und alle zugehörigen Dateien. Mit einem *öffentlichen Schlüssel*, einem *privaten Schlüssel* und dem Zertifikat kann *Fishhead Software* nun das Tool *jar* dazu verwenden, die Archivdatei mit einer digitalen Signatur zu versehen.
5. *Fishhead Software* platziert das gekennzeichnete Archiv so auf der Website, daß es sich zusammen mit dem *öffentlichen Schlüssel* herunterladen läßt.

Damit hat *Fishhead Software* das Spiel in einer Weise zur Verfügung gestellt, die es den Benutzern

ermöglicht, seine Sicherheitsbedenken auszuräumen. Eine dieser Personen, die sich dazu entschlossen haben, das Applet-Spiel von *Fishhead* zu laden, ist der Web-Benutzer *Gilbert*. Er arbeitet mit einem Browser, der Java 1.2 unterstützt.

Seine Aktionen sind einfacher:

1. *Gilbert* erkennt, daß das neue Applet-Spiel von *Fishhead* Risiken birgt und er möchte zunächst wissen, ob es sich bei dem Hersteller um eine vertrauenswürdige Firma handelt. Er lädt den *öffentlichen Schlüssel* von *Fishhead* herunter.
2. Er beschließt, daß es sich bei *Fishhead Software* um eine vertrauenswürdige Firma handelt und verwendet *jarsigner* zusammen mit dem *öffentlichen Schlüssel* dazu, die Firma in seine Systemliste für sichere Hersteller aufzunehmen.

Gilbert kann das Applet-Spiel nach Herzenslust spielen. Dieses Spiel kann alle Funktionen ausführen, die andernfalls nur Java-Anwendungen vorbehalten sind. Das heißt: Selbstverständlich kann dieses Spiel immer noch alle nur denkbaren Schäden auf seinem System anrichten, ebenso wie jede andere Software auch, die auf einem Computer installiert wird. Der Vorteil der digitalen Signatur besteht aber darin, daß der Benutzer weiß, welcher Programmierer das Spiel erstellt hat. Programmierer, die Viren verteilen möchten, werden wohl kaum eine Spur hinterlassen, die direkt zu ihrem Haus weist.

Vielleicht fragen Sie sich, warum es einen öffentlichen und einen privaten Schlüssel gibt? Wenn nur beide Schlüssel zusammen eine Firma identifizieren können, wie ist es dann möglich, daß der öffentliche Schlüssel allein zur Identifikation von *Fishhead* dient?

Ein öffentlicher und ein privater Schlüssel ergeben zusammen ein Sicherheitssystem. Da nur beide zusammen *Fishhead* Software ganz identifizieren, sind beide Schlüssel nur der Firma selbst zugänglich, andernfalls könnte sich jemand anders als *Fishhead* ausgeben und niemand könnte dies enthüllen. Indem *Fishhead* seinen privaten Schlüssel schützt, schützt es seine Identität und seine Reputation.

Die Prüfung, die *Signatures ,R` US* unter Verwendung des öffentlichen Schlüssels durchführt, stellt vor allem sicher, daß der öffentliche Schlüssel wirklich zu dieser Firma gehört. Da der öffentliche Schlüssel an jeden weitergegeben werden kann, stellt *Fishhead* diesen auf der Website zur Verfügung. Als Teil des Zertifizierungsvorgangs kann *Signatures ,R` US* diesen herunterladen und mit jenem vergleichen, den es erhalten hat. Die Zertifizierungsgruppe agiert als eine Art Ersatz für den privaten Schlüssel und stellt sicher, daß der öffentliche Schlüssel legitim ist. Das herausgegebene Zertifikat ist mit dem öffentlichen Schlüssel verbunden, der nur zusammen mit dem privaten Schlüssel von *Fishhead* verwendet werden kann.

Jeder, der mit dem Tool *keytool* arbeitet, kann ein Zertifikat für einen öffentlichen Schlüssel herausgeben. *Fishhead* könnte sich das Zertifikat auch selbst ausstellen. Doch mit einem solchen Schritt würde die Glaubwürdigkeit deutlich herabgesetzt. Eine bekannte, unabhängige Zertifizierungsgruppe sorgt für Vertrauenswürdigkeit.

Alle Elemente zusammen bilden eine verlässliche digitale Signatur für ein Java-Archiv. Die JavaSoft-Dokumentation zu *keytool*, der Applet-Kennzeichnung *jar* und andere neue Sicherheitsfunktionen stehen unter folgender Web-Adresse zur Verfügung:

<http://www.javasoft.com/products/JDK/1.2/docs/guide/security>

Browserspezifische Signaturen

Zu dem Zeitpunkt, als dieses Buch entstand, war die einzige Möglichkeit, ein Applet zu signieren, die Methoden anzuwenden, die die Entwickler bei Netscape und Microsoft für deren eigene Browser entwickelt haben. Sie müssen deren Tools verwenden und ein Applet mit beiden Methoden signieren, wenn Sie die Benutzer beider Browser erreichen wollen.

Um ein Applet für den Microsoft Internet Explorer zu signieren, ist folgendes nötig:

- Eine digitale Microsoft-Authenticode-ID von einer Firma, z.B. wie VeriSign oder Thawte - Firmen, die Ihre Identität verifizieren.
- Internet Explorer 4.0 oder höher.
- Die folgenden Tools aus dem Microsoft-Java-Software-Development-Kit: cabarc.exe , chktrust.exe, signcode.exe und die .DLL-Dateien javasign.dll und signer.dll. Dieses Kit ist bei Microsoft zum Download verfügbar unter der Adresse <http://www.microsoft.com/java/download.htm>.

Um ein Applet für den Netscape Navigator zu signieren, ist folgendes nötig:

- Eine digitale Netscape-Object-Signing-Software-Publishing-ID, die bei einer der Firmen, die in der Liste auf der Webseite unter <https://certs.netscape.com/client.html> aufgeführt sind.
- Das Netscape-Signing-Tool. Dieses steht auf der Webseite mit der Adresse <http://developer.netscape.com/software/signedobj/jarpack.html> zur Verfügung. Das Signing-Tool bietet die Möglichkeit, ein Testzertifikat zu verwenden, bevor Sie eine digitale ID beziehen müssen.



Dokumentationsmaterial zur Anwendung dieser Tools finden Sie auf den Websites, von denen Sie diese Tools herunterladen. Zusätzlich hat Daniel Griscom von Sutable Systems eine exzellente Informationsquelle für das Signieren von Java-Code unter der folgenden Web-Adresse zusammengestellt: http://www.suitable.com/Doc_CodeSigning.shtml

Sicherheitsrichtlinien

Vor Java 1.2 ging man davon aus, daß alle Applikationen als gänzlich vertrauenswürdig eingestuft werden und in der Lage sein sollten, alle Features der Sprache zu verwenden - dies war auch entsprechend in die Sprache integriert.

Um es einfacher zu gestalten, Applikationen zu erstellen, die in ihren Möglichkeiten eingeschränkter sind, werden Applikationen inzwischen derselben Sicherheitsüberprüfung wie Applets unterzogen.

In der allgemeinen Praxis ändert dies nichts daran, wie Applikationen geschrieben und ausgeführt werden - die Applikationen, die Sie im Laufe des Buches erstellt haben, sollten keine sicherheitsbezogenen Ausnahmen verursacht haben, als Sie diese auf Ihrem System ausgeführt haben. Der Grund dafür ist, daß die Sicherheitsrichtlinien bei der Installation des JDK so liberal wie möglich angelegt wurden, so daß es der Applikation möglich ist, alle Features zu nutzen.

Die Sicherheitsrichtlinien werden in der Datei java.policy gespeichert. Diese Datei finden Sie in dem Unterordner lib\security\ des Hauptordners der JDK-Installation. Diese Datei kann mit jedem Texteditor bearbeitet werden, obwohl Sie diese nicht verändern sollten, solange Sie nicht über die Funktionsweise gut Bescheid wissen. Sie können auch ein Tool mit grafischer Oberfläche zur Bearbeitung der Richtlinien verwenden. Dieses trägt den Namen policytool.

Einen Überblick über die Sicherheitsfeatures, die in Java 1.2 implementiert wurden, steht bei JavaSoft auf der folgenden Webseite zur Verfügung:

<http://java.sun.com/products/jdk/1.2/docs/guide/security/spec/security-spec.doc.html>

Zusammenfassung

Heute haben Sie das Arbeiten mit Ausnahmen gelernt und erfahren, daß Sie damit Robustheit in Ihrem Programmdesign erreichen können.

Sie haben eine Fülle von Ausnahmen kennengelernt, die in der Java-Klassenbibliothek definiert sind. Sie haben die Verwendung von throw, try und catch gelernt, um mögliche Fehler und Ausnahmen zu handhaben. Ferner haben Sie gesehen, daß die ausschließliche Arbeit mit der Java-Ausnahmenhandhabung dem Programmierer einige Einschränkungen auferlegt. Sie haben aber auch erfahren, daß diese Einschränkungen in anderer Hinsicht reich belohnt werden.

Schließlich haben Sie die finally-Klausel gelernt, mit der Sie sicher sein können, etwas zu erreichen, egal was.

Außerdem haben Sie heute die Grundlagen gelernt, wie das Sicherheitsmodell von Java 1.2 implementiert ist, und welche Möglichkeiten die verschiedenen Browserhersteller anbieten, um die normalen Sicherheitseinschränkungen von Applets mit digitalen Signaturen zu umgehen.

Fragen und Antworten

Frage:
Ich habe den Unterschied zwischen Exceptions, Errors und RuntimeExceptions noch nicht verstanden. Könnten Sie das nochmal auf andere Weise erklären?

Antwort:
Errors werden durch Probleme beim dynamischen Linken oder der virtuellen Maschine verursacht und sind deshalb für die meisten Programme auf einer zu niedrigen Ebene, um sie behandeln zu können (obwohl ausgefeilte Entwicklungsbibliotheken und Umgebungen wahrscheinlich dazu in der Lage sind). RuntimeExceptions werden durch die normale Ausführung des Java-Codes erzeugt. Sie spiegeln zwar gelegentlich eine Bedingung wider, die explizit gehandhabt werden soll oder kann, sind aber meist auf einen Programmierfehler zurückzuführen und sollen lediglich eine Fehlermeldung ausgeben, damit der Fehler abgegrenzt werden kann. Exceptions, die nicht zu RuntimeExceptions gehören (z.B. IOExceptions), sind Bedingungen, die aufgrund ihrer Natur explizit durch einen robusten und gut durchdachten Code gehandhabt werden sollten. Die Java-Klassenbibliothek enthält einige davon, die extrem wichtig sind, um das System sicher und korrekt benutzen zu können. Der Compiler unterstützt Sie bei der Handhabung dieser Ausnahmen durch Kontrollen und Einschränkungen über die throws-Klausel.

Frage:
Gibt es eine Möglichkeit, die straffen Einschränkungen, denen Methoden durch die throws-Klausel unterliegen, irgendwie zu umgehen?

Antwort:
Ja, die gibt es. Nehmen wir an, Sie haben lange genug nachgedacht und sind fest entschlossen, diese Einschränkung zu umgehen. Das ist fast nie der Fall, weil die richtige Lösung die ist, Ihre Methoden neu auszulegen, um die auszuwerfenden Ausnahmen entsprechend zu berücksichtigen. Wir stellen uns aber vor, daß Sie durch eine Systemklasse aus irgendeinem Grund in einer Zwangsjacke stecken. Ihre erste Lösung ist, von RuntimeException eine Subklasse für Ihre spezielle Ausnahme zu erstellen. Sie können damit nach Herzenslust Ausnahmen auswerfen, denn die throws-Klausel, die Sie belästigt hat, muß diese neuen Ausnahmen nicht enthalten. Müssen Sie viele solcher Ausnahmen unterbringen, besteht ein eleganter Ansatz darin, einige neue Ausnahmenschnittstellen in Ihre neuen Runtime-Klassen beizumischen. Welche dieser neuen Schnittstellen Sie mit catch auffangen wollen, steht Ihnen frei (keine der normalen Runtime-Ausnahmen muß aufgefangen werden), während eventuelle Überbleibsel der (neuen)

Runtime-Ausnahmen (absolut zulässig) diese andernfalls lästige Standardmethode der Bibliothek durchlaufen können.

Frage:

Ich finde es zuweilen ganz schön lästig, Ausnahmebedingungen zu handhaben. Was kann mich eigentlich davon abhalten, mich durch eine Methode mit einer throws-Klausel abzuschotten?

Beispielsweise so:

```
try { that AnnoyingMethod(); } catch (Throwable t) { }
```

Damit könnte ich doch alle Ausnahmen einfach ignorieren, oder?

Antwort:

Sie würden damit nichts außer Ihrem Gewissen abschotten. In manchen Fällen sollen Sie nichts unternehmen, weil das einfach für die Implementierung der betreffenden Methode das Richtige ist. In anderen Fällen müssen Sie sich aber wohl oder übel durch diese lästigen Prozeduren durchkämpfen. Sie gewinnen dadurch ja auch mehr Erfahrung. Auch der beste Programmierer sollte sich vor solchen öden Dingen nicht scheuen, zumal er reichlich dafür belohnt wird.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Tag 17

Java und Streams

Heute lernen Sie alles über Java-Datenstreams mit folgenden Schwerpunkten:

- *Eingabedatenstreams*: wie sie erstellt und benutzt und wie ihr Ende erkannt werden kann; wie man sie filtert und verschachtelt
- *Ausgabedatenstreams*: wie oben, jedoch betreffen sie Ausgaben
- *Reader*: wie sie erstellt, benutzt und gefiltert werden, und wozu es nützlich ist, sie zu verschachteln
- *Writer*: wie Reader, jedoch betreffen sie Datenstreams in umgekehrter Richtung

Sie lernen auch zwei Datenstreamschnittstellen kennen, die das Lesen und Schreiben getippter Datenstreams vereinfachen. Ferner lernen Sie, wie ganze Objekte gelesen und geschrieben werden. Und Sie lernen mehrere Utility-Klassen kennen, die benutzt werden, um auf das Dateisystem zuzugreifen. Wir beginnen mit einer kurzen Geschichte über Datenstreams.

Eine der ersten Erfindungen des Unix-Betriebssystems war die *Pipe*. Eine *Pipe* ist ein nichtinterpretierter Byte-Stream, der zur Kommunikation zwischen Programmen (bzw. »gegabelten« Kopien eines Programms) oder zum Lesen und Schreiben von verschiedenen Peripheriegeräten und Dateien benutzt wird. Durch Vereinheitlichung aller möglichen Kommunikationsarten in einer einzigen Metapher ebnete Unix den Weg für eine ganze Reihe ähnlicher Neuerungen, die schließlich in der Abstraktion namens *Streams* oder *Datenstreams* gipfelten.



Ein *Stream* oder *Datenstream* ist ein Kommunikationspfad zwischen der Quelle und dem Ziel eines Informationsblocks.

Dieser Informationsblock, d.h. ein nichtinterpretierter Byte-Stream, kann von jeder »Pipe-Quelle«, dem Rechnerspeicher oder auch vom Internet kommen. Quelle und Ziel eines Datenstreams sind willkürliche Erzeuger bzw. Verbraucher von Bytes. Darin liegt die Leistung dieser Abstraktion. Sie müssen beim Lesen nichts über die Quelle und beim Schreiben nichts über das Ziel des Datenstreams wissen.

Allgemeine Methoden, die von jeder beliebigen Quelle lesen können, akzeptieren ein Streamargument, das die Quelle bezeichnet. Allgemeine Methoden zum Schreiben akzeptieren einen Stream, um das Ziel zu bestimmen. Arbiträre *Prozessoren* (oder *Filter*) haben zwei Streamargumente. Sie lesen vom ersten, verarbeiten die Daten und schreiben die Ergebnisse in den zweiten. Diese Prozessoren kennen weder Quelle noch Ziel der Daten, die sie verarbeiten. Quelle und Ziel können sehr unterschiedlich sein: von zwei Speicherpuffern auf dem gleichen lokalen Rechner über ELF-Übertragungen von und zu einer Unterwasserstation bis zu Echtzeit-Datenstreams einer NASA-Sonde im Weltraum.

Durch Entkoppeln des Verbrauchs, der Verarbeitung und der Produktion der Daten von Quelle und Ziel dieser Daten können Sie jede beliebige Kombination mischen, während Sie Ihr Programm schreiben. Künftig, wenn neue, bisher nicht bekannte Formen von Quelle oder Ziel (oder Verbraucher, Verarbeitung und Erzeuger) erscheinen, können sie im gleichen Rahmen ohne Änderung von Klassen benutzt werden. Neue Streamabstraktionen, die höhere Interpretationsebenen »oberhalb« der Bytes unterstützen, können völlig unabhängig von den zugrundeliegenden Mechanismen für den Transport der Bytes geschrieben werden.



Eine solche Interpretation der höheren Ebene hat genau das bewirkt: Die mit Java 1.1 eingeführten Streams zur Objektserialisierung werden »oberhalb« des Datenstreammechanismus von Version 1.0 geschrieben. Sie lernen noch in der heutigen Lektion mehr über die *Serialisation* von Objekten.

Die Basis dieses Streamgerüsts bilden die zwei abstrakten Klassen *InputStream* und *OutputStream*. Wenn Sie sich ein Hierarchiediagramm von *java.io* ansehen, erkennen Sie, daß unter diesen Klassen eine Fülle von Klassen steht, die den

breiten Bereich von Datenstreams im System, aber auch eine äußerst gut ausgelegte Hierarchie von Beziehungen zwischen diesen Datenstreams aufzeigt. Ein ähnlicher Baum ist im Diagramm von java.io-rw vorhanden, der in den abstrakten Eltern Reader und Writer seine Wurzeln hat. Wir beginnen mit diesen Elternklassen und arbeiten uns durch diesen buschigen Baum.



Da sich jede Klasse der heutigen Lektion im Paket java.io befindet, müssen Sie die einzelnen Klassen vor der Verwendung entweder importieren (oder ihren ausgeschriebenen Namen, z.B. java.io.InputStream benutzen) oder am Beginn der Klasse eine Anweisung import java.io.* schreiben. Alle Methoden der heutigen Lektion sind so deklariert, daß sie Ausnahmen vom Typ IOExceptions auswerfen können. Diese neue Unterklasse von Exception verkörpert konzeptionell alle möglichen Ein- und Ausgabefehler, die bei der Benutzung von Datenstreams, Reader und Writer usw. in Java vorkommen können. (IOException hat viele Subklassen, die spezifischere Ausnahmen definieren, die ebenfalls ausgeworfen werden können.) Vorläufig genügt zu wissen, daß Sie eine IOException entweder mit catch auffangen oder in eine Methode stellen müssen, die sie weitergeben kann. (Zur Erinnerung: Mit Ausnahmen haben wir uns in der 16. Lektion beschäftigt.)

Eingabedatenstreams und Reader

Die Grundlagen für alle Eingabeoperationen von Java bilden die zwei in den nächsten Unterabschnitten beschriebenen Klassen. Nach deren Definition ergeben sich die analogen Klassen, die aus den hier dargestellten stammen, weil diese Klassenpaare fast identische Methodenschnittstellen haben und auf die gleiche Weise benutzt werden.

Die abstrakten Klassen InputStream und Reader

InputStream ist eine abstrakte Klasse, die die Grundlagen für das Lesen eines *Byte- Streams* durch den Verbraucher (das Ziel) von einer Quelle definiert. Die Identität der Quelle und die Art, wie die Bytes erstellt und befördert werden, ist nicht relevant. Bei der Verwendung eines Eingabestreams sind *sie* das Ziel dieser Bytes. Das ist alles, was Sie wissen müssen.

Reader ist eine abstrakte Klasse, die die Grundlagen definiert, wie ein Ziel (Verbraucher) einen aus *Zeichen* bestehenden und von irgendeiner Quelle kommenden Datenstream liest. Der Leser (Reader) und alle seine Unterklassen sind analog der Klasse InputStream und allen ihren Unterklassen, ausgenommen, daß sie als zugrundeliegende Informationseinheiten Zeichen anstelle von Bytes benutzen.

read()

Die wichtigste Methode für den Verbraucher eines Eingabestreams (oder Reader) ist die, die die Bytes (Zeichen) von der Quelle liest. Diese Methode ist read(). Sie existiert in vielen Varianten, von denen in der heutigen Lektion je ein Beispiel aufgezeigt wird.

Jede dieser read()-Methoden ist so definiert, daß sie warten muß, bis alle angeforderten Eingaben verfügbar sind. Sorgen Sie sich nicht wegen dieser Einschränkung. Dank Multithreading können Sie viele andere Dinge realisieren, während ein Thread auf eine Eingabe wartet. Üblicherweise wird ein Thread je einem Eingabestream (und je einem Ausgabestream) zugewiesen, der allein für das Lesen vom (oder Schreiben zum) jeweiligen Stream zuständig ist. Die Eingabe-Threads können dann die Informationen zur Verarbeitung an andere Threads abgeben. Dadurch überlappt natürlich die I/O- Zeit Ihres Programms mit seiner Berechnungszeit.

Hier die erste Form von read():

```
InputStream s      = getAnInputStreamFromSomewhere();
Reader      r      = getAReaderFromSomewhere();
byte[]      bbuffer = new byte[1024];    // Kann beliebige Größe sein
char[]      cbuffer = new char[1024];

if (s.read(bbuffer) != bbuf.length || r.read(cbuffer) != cbuf.length)
    System.out.println("Ich habe weniger erhalten als erwartet.");
```



Sofern nicht anders angegeben, wird jede Methode in den im folgenden beschriebenen Datenstream-, Reader- und Writer-Klassen auf die gleiche Weise benutzt wie die Klasse dieses Abschnitts. So wird beispielsweise die obige `read()`-Methode sowohl in `InputStream` als auch in `Reader` gleich benutzt. Hier und in der gesamten Lektion gehen wir davon aus, daß entweder `import java.io.*` vor jedem Beispiel erscheint oder daß Sie alle Referenzen auf `java.io`-Klassen mit dem Präfix `java.io` schreiben.

Diese Form von `read()` versucht, den gesamten zugeteilten Puffer zu füllen. Gelingt es ihr nicht (normalerweise, weil das Ende des Eingabestreams vorher erreicht wird), gibt sie die tatsächliche Anzahl von Bytes aus, die in den Puffer eingelesen wurden. Danach gibt ein eventueller weiterer Aufruf von `read()` den Wert `-1` zurück, was anzeigt, daß das Ende des Datenstreams erreicht ist. Die `if`-Anweisung funktioniert auch, wenn der Datenstream leer ist, weil `-1` nie der Pufferlänge entspricht.



Im Gegensatz zu C wird der Fall `-1` in Java nicht benutzt, um einen Fehler anzuzeigen. Eventuelle I/O-Fehler werfen Instanzen von `IOException` aus (was wir noch nicht mit `catch` auffangen). Sie haben gestern gelernt, daß alle bestimmten Werte durch Ausnahmen ersetzt werden können und sollten. Im letzten Beispiel ist `-1` ein historischer Anachronismus. Sie werden gleich einen besseren Ansatz zum Anzeigen des Streamendes mit der Klasse `DataInputStream` kennenlernen.

Sie können auch in einen Bereich Ihres Puffers einlesen, indem Sie den Versatz (*Offset*) und die gewünschte Länge als Argumente in der zweiten Form von `read()` angeben:

```
s.read(bbuffer, 100, 300);
r.read(cbuffer, 100, 300);
```

Bei diesem Beispiel werden die Bytes (Zeichen) 100 bis 399 gelesen. Ansonsten verhält es sich genauso wie mit der vorherigen `read()`-Methode. In der aktuellen Version benutzt die Standardimplementierung der ersten Form von `read()` die zweite Alternative:

```
public int  read(byte[]  b) throws IOException {           /* Von InputStream.java */
    return  read(b, 0, b.length);
}
public int  read(char[]  cbuf) throws IOException {        /* Von Reader.java */
    return  read(cbuf, 0, cbuf.length);
}
```

In der dritten Form können die Bytes (Zeichen) auch einzeln eingelesen werden:

```
InputStream s = getAnInputStreamFromSomewhere();
InputStream r = getAReaderFromSomewhere();
byte        b;
char        c;
int         byteOrMinus1, charOrMinus1;

while ((byteOrMinus1 = s.read()) != -1 && (charOrMinus1 = r.read()) != -1) {
    b = (byte) byteOrMinus1;          c = (char) charOrMinus1;
    . . .      // Verarbeite Byte b (oder Zeichen c)
}
. . .      // Datenstreamende erreicht
```



Aufgrund der allgemeinen Vorliebe für Integer in Java und weil die `read()`-Methode in diesem Fall einen `int` zurückgibt, kann die Verwendung des Typs `byte` (oder `char`) im Code ein bißchen frustrierend sein. Man muß ständig das Ergebnis von arithmetischen Ausdrücken oder `int`-Ausgabewerten in die gewünschte Größe umwandeln. Da `read()` in diesem Fall eigentlich `byte` (oder `char`) zurückgeben sollte, halte ich es für besser, die Methode als solche zu deklarieren und zu

verwenden. In Fällen, in denen man das Gefühl hat, der Bereich einer Variablen ist auf byte, char oder short begrenzt, sollte man sich die Zeit nehmen, dies nicht mit int, sondern als was es ist zu deklarieren. Nebenbei bemerkt, speichert ein Großteil des Codes der Java-Klassenbibliothek das Ergebnis von read() als int. Das zeugt von der Menschlichkeit des Java- Teams - *jeder* kann schließlich Fehler machen.

skip()

Für den Fall, daß Sie einige Bytes in einem Datenstream überspringen oder von einer anderen Stelle mit dem Lesen des Datenstreams beginnen wollen, gibt es eine mit read() vergleichbare Methode:

```
if (s.skip(1024) != 1024 || r.skip(1024) != 1024)
    System.out.println("Ich habe weniger übersprungen als erwartet.");
```

Dadurch werden die nächsten 1024 Byte des Eingabestreams übersprungen. skip() nimmt und gibt einen long-Integer zurück, weil Datenstreams nicht auf eine bestimmte Größe begrenzt werden müssen. Die Standardimplementierung von skip() in InputStream benutzt einfach read():

```
public long skip(long n) throws IOException {          /* Von InputStream.java */
    byte[] data = new byte[(int) n & 0xFFFFFFFF];
    return read(data);
}
```



Diese Implementierung unterstützt große skip-Methoden nicht korrekt, weil ihr long- Argument in eine Ganzzahl (int) umgewandelt wird. (Bei der Implementierung von skip() in Reader läuft das korrekt ab - lange Zeichenzahlen werden übersprungen.) In Subklassen muß diese Standardimplementierung überladen werden, damit dies richtig abgearbeitet wird. Das ist nicht so einfach, wie Sie denken, weil Java keine Ganzzahlentypen als Array-Indizes zuläßt, die größer sind als int.

available() und ready()

Wenn Sie wissen wollen, wie viele Bytes ein Datenstream momentan umfaßt (oder ob beim Leser weitere Zeichen auf Sie warten), können Sie so fragen:

```
if (s.available() < 1024)
    System.out.println("Momentan ist zu wenig verfügbar.");
if (r.ready() != true)
    System.out.println("Momentan sind keine Zeichen verfügbar.");
```

Dadurch wird Ihnen die Anzahl von Bytes mitgeteilt, die ohne Blockierung gelesen werden können (oder ob Sie irgendwelche Zeichen lesen können). Aufgrund der abstrakten Natur der Quelle dieser Bytes sind Datenstreams eventuell nicht in der Lage, Ihnen auf diese Frage eine Antwort zu geben. Einige Datenstreams geben beispielsweise immer 0 (oder false) zurück. Dieser Wert ist der Rückgabewert der Standardimplementierung von available() (oder ready()).

Sofern Sie keine spezifischen Unterklassen von InputStream verwenden, die Ihnen eine vernünftige Antwort auf diese Frage geben, sollten Sie sich nicht auf diese Methode verlassen. Multithreading schließt ohnehin viele Probleme in Verbindung mit der Blockierung während der Wartezeit auf einen Datenstream aus. Damit schwindet einer der vorrangigen Nutzen von available() (oder ready()) dahin.

mark() und reset()

Einige Datenstreams unterstützen die Markierung einer Position im Datenstream und das spätere Zurücksetzen des Datenstreams auf diese Position, um die Bytes (Zeichen) ab dieser Stelle erneut zu lesen. Der Datenstream müßte sich dabei an alle Bytes (Zeichen) »erinnern«, deshalb gibt es eine Einschränkung, in welchem Abstand in einem Datenstream markiert und zurückgesetzt werden kann. Ferner gibt es eine Methode, die fragt, ob der Datenstream dies überhaupt unterstützt. Hier ein Beispiel:

```
InputStream s = getAnInputStreamFromSomewhere();
Reader      r = getAReaderFromSomewhere();

if (s.markSupported() && r.markSupported()) {    // Markieren
                                                    // unterstützt?
```



```

        . . .                                // Datenstream eine Weile
                                           // lesen

        s.mark(1024);      r.mark(1024);

        . . .                                // Weniger als 1024 weitere
                                           // Bytes (Zeichen)
                                           // lesen

        s.reset();        r.reset();

        . . .                                // Wir können diese Bytes
                                           // (Zeichen) jetzt erneut
                                           // lesen

    } else {

        . . .                                // Nein, führe irgendeine
                                           // Alternative aus

    }

```

Durch Markieren eines Datenstreams wird die Höchstzahl der Bytes (Zeichen) bestimmt, die vor dem Zurücksetzen weitergegeben werden soll. Dadurch kann der Datenstream den Umfang seines »Speichers« eingrenzen. Läuft diese Zahl durch, ohne daß ein `reset()` erfolgt, wird die Markierung ungültig und der Versuch zurückzusetzen erzeugt eine Ausnahme.

Markieren und Zurücksetzen eines Datenstreams ist nützlich, wenn der Streamtyp (oder der nächste Streamteil) identifiziert werden soll. Hierfür verbrauchen Sie aber einen beträchtlichen Anteil davon im Prozeß. Oft liegt das daran, daß man mehrere Parser hat, denen man den Datenstream übergeben kann. Sie verbrauchen aber eine (Ihnen unbekannte) Zahl an Bytes (Zeichen), bevor sie sich entscheiden, ob der Datenstream ihr Typ ist. Setzen Sie eine große Größe als Lesegrenze, und lassen Sie jeden Parser ablaufen, bis er entweder einen Fehler ausgibt oder die Syntaxanalyse erfolgreich beendet. Wird ein Fehler ausgegeben, setzen Sie ihn zurück, und versuchen Sie es mit dem nächsten Parser.



Die Standardimplementierung von `markSupported()` gibt zwar `false` zurück und `reset()` wirft eine `IOException` aus, aber sowohl bei `InputStream` als auch bei `Reader` macht `mark()` von `InputStream` nichts, während `mark()` von `Reader` eine `IOException` erzeugt. Das bricht (leider) die fast perfekte Symmetrie der beiden Klassen.

close()

Da Sie nicht wissen, welche Ressourcen ein offener Datenstream darstellt und wie diese Ressourcen zu behandeln sind, nachdem der Datenstream gelesen wurde, müssen Sie einen Datenstream normalerweise explizit schließen, damit er diese Ressourcen freigeben kann. Selbstverständlich können das der Garbage-Collector und eine Methode `finalize()` ebenfalls erledigen. Es könnte aber sein, daß Sie den Datenstream erneut öffnen müssen, bevor die Ressourcen dieses asynchronen Prozesses freigegeben werden. Bestenfalls ist das ärgerlich oder verwirrend. Im schlechtesten Fall entsteht ein unerwarteter, schwer auszumachender Fehler. Da Sie hierbei mit der Außenwelt, d.h. mit externen Ressourcen, zu tun haben, ist es ratsam, genau anzugeben, wann deren Benutzung enden soll:

```

InputStream  s = alwaysMakesANewInputStream();
Reader       r = alwaysMakesANewReader();

try {
    . . .      // Benutze s (oder r) nach Herzenslust
} finally {
    s.close();  r.close();
}

```

Gewöhnen Sie sich an die Verwendung von `finally`. Sie stellen damit sicher, daß Aktionen (z.B. das Schließen eines Datenstreams) auf jeden Fall ausgeführt werden. Selbstverständlich gehen Sie davon aus, daß der Datenstream immer erfolgreich erzeugt wird. Ist das nicht stets der Fall und wird zuweilen null zurückgegeben, gehen Sie auf Nummer Sicher:

```

InputStream  s = tryToMakeANewInputStream();
Reader       r = tryToMakeAReader();

if (s != null && r != null) {
    try {

```

```

        } finally {
            s.close();    r.close();
        }
    }
}

```

Alle Eingabestreams stammen von der abstrakten Klasse `InputStream`, und alle Reader stammen von `Reader` ab. Alle haben die bisher beschriebenen Methoden. Somit könnte `InputStream s` (oder `Reader r`) im vorherigen Beispiel auch einen der komplexeren Eingabestreams haben, die in den nächsten Abschnitten beschrieben werden.



Konkrete Subklassen von `InputStream` brauchen nur die dritte Form von `read()` ohne Argumente zu implementieren, um alle übrigen Methoden zum Arbeiten zu bringen (`InputStream` hat in der Standardimplementierung die Methode `close()`, die nichts bewirkt). Unterklassen von `Reader` müssen aber sowohl `close()` als auch die zweite Form von `read()` mit den drei Argumenten implementieren.

ByteArrayInputStream und CharArrayReader

Durch »Umkehr« einiger der vorherigen Beispiele mit `read()` würde man einen Eingabestream (oder Reader) aus einem Byte- oder Zeichenarray erstellen. Genau das besorgt `ByteArrayInputStream` (bzw. `CharArrayReader`):

```

byte[]  bbuffer = new byte[1024];
char[]  cbuffer = new char[1024];

fillWithUsefulData(bbuffer);    fillWithUsefulData(cbuffer);

InputStream  s = new ByteArrayInputStream(bbuffer);
Reader      r = new CharArrayReader(cbuffer);

```

Reader des neuen Datenstreams `s` (`r`) sehen einen Datenstream mit einer Länge von 1024 Byte (Zeichen), d.h. den Inhalt des Arrays `bbuffer` (`cbuffer`). Der Konstruktor dieser Klasse hat wie `read()` einen Versatz (*Offset*) und eine Länge:

```

InputStream  s = new ByteArrayInputStream(bbuffer, 100, 300);
Reader      r = new CharArrayReader(cbuffer, 100, 300);

```

Hier ist der Datenstream 300 Byte (Zeichen) lang und enthält Bytes 100-399 aus dem Array `bbuffer` (`cbuffer`).



Damit haben Sie die ersten Beispiele des *Erstellens* von Eingabestreams gesehen. Diese neuen Datenstreams werden an die einfachsten aller möglichen Datenquellen angehängt - an ein Byte- oder Zeichenarray im Speicher des lokalen Rechners.

`ByteArrayInputStream` (`CharArrayReader`) implementiert lediglich die Standardmethoden wie alle Eingabestreams. Hier hat die Methode `available()` (`ready()`) aber eine ganz bestimmte Aufgabe: Sie gibt 1024 bzw. 300 (`true` und `true`) für die zwei Instanzen von `ByteArrayInputStream` (`CharArrayReader`) zurück, die Sie zuvor erstellt haben, weil sie genau weiß, wie viele Bytes (Zeichen) verfügbar sind. Auch `markSupported()` gibt `true` zurück. Schließlich wird der Datenstream durch Aktivieren von `reset()` ohne ein vorangehendes `mark()` an den Anfang seines Puffers zurückgesetzt.

FileInputStream und FileReader

Eine der häufigsten Verwendungen von Datenstreams und historisch die älteste ist das Anhängen von Datenstreams an Dateien im Dateisystem. Hier wird beispielsweise ein solcher Eingabestream (oder Reader) auf einem Unix-System erstellt:

```

InputStream  s = new FileInputStream("/Irgendein/Pfad/und/Dateiname");
Reader      r = new FileReader("/Irgendein/Pfad/und/Dateiname.utf8");

```



Applets, die versuchen, solche Datenstreams im Dateisystem zu öffnen, zu lesen oder zu schreiben, können in den meisten Browsern Sicherheitsverletzungen verursachen. (Bei einigen Browsern kann der Benutzer verschiedene Sicherheitsfunktionen in bezug auf das Lesen und Schreiben in Verzeichnissen einstellen.) Wie Sie gestern erfahren haben, besteht in Java 1.2 die Möglichkeit, durch das Signieren von Applets diesen den Zugriff auf das Dateisystem zu gestatten.

Sie können Datenstreams auch aus einem zuvor aktivierten `FileDescriptor` oder einer Datei (`File`) erstellen:

```
InputStream s = new FileInputStream(FileDescriptor.in); /* Standardeingabe */
Reader      r = new FileReader(FileDescriptor.in);
InputStream s = new FileInputStream(new File("/Irgendein/Pfad/und/Dateiname"));
Reader      r = new FileReader(new File("/Irgendein/Pfad/und/Dateiname.utf8"));
```

Da dies auf einer tatsächlichen Datei mit einer bestimmten Länge basiert, kann der erzeugte Eingabestream (`Reader`) in allen drei Fällen problemlos `available()` (`ready()`) und `skip()` implementieren (wie übrigens auch `ByteArrayInputStream` und `CharArrayReader`).

`FileReader` ist eigentlich eine triviale Unterklasse der weiteren `Reader`-Klasse `InputStreamReader`, die jeden beliebigen Eingabestream (`InputStream`) kapseln und ihn in einen Zeichenleser umwandeln kann. Somit besteht die Implementierung von `FileReader` lediglich aus der Aufforderung von `InputStreamReader`, (selbst) einen `FileInputStream` zu kapseln:

```
public class FileReader extends InputStreamReader { /* Von FileReader.java */
    public FileReader(String fileName) throws FileNotFoundException {
        super(new FileInputStream(fileName))
    }
    public FileReader(File file) throws FileNotFoundException {
        super(new FileInputStream(file))
    }
    public FileReader(FileDescriptor fd) throws FileNotFoundException {
        super(new FileInputStream(fd))
    }
}
```

`FileInputStream` (nicht aber `FileReader`) kennt darüber hinaus noch ein paar Tricks:

```
FileInputStream aFIS = new FileInputStream("EinDateiname");
```

```
FileDescriptor myFD = aFIS.getFD();
/* aFIS.finalize(); */ // Aktiviert close(), wenn GC automatisch aufgerufen wird
```



Um eine `getFD()`-Methode aufzurufen, müssen Sie die Datenstream-Variable `aFIS` als `FileInputStream`-Typ deklarieren, weil ein einfacher Eingabestream (`InputStream`) von `getFD()` keine Ahnung hat.

Ein Aspekt ist ganz klar: `getFD()` gibt den Bezeichner der Datei zurück, auf der der Datenstream basiert. Der zweite Aspekt ist eine interessante Kurzform, mit der Sie beliebige `FileInputStream` erstellen können, ohne sich um deren spätere Schließung Gedanken machen zu müssen. Die Implementierung von `finalize()` (einer geschützten Methode) durch `FileInputStream` schließt den Datenstream. Im Gegensatz zum vorherigen Beispiel sollten Sie eine `finalize()`-Methode nie direkt aufrufen. Der Garbage-Collector ruft sie auf, nachdem er festgestellt hat, daß der Datenstream nicht mehr gebraucht wird. Das System schließt den Datenstream (irgendwann).

Sie können sich diese Lässigkeit leisten, weil Datenstreams, die auf Dateien basieren, nur wenige Ressourcen binden. Diese Ressourcen können nicht versehentlich vor ihrer Beseitigung durch den Garbage-Collector (entgegen den vorherigen Beispielen mit `finalize()` und `close()`) wiederverwendet werden. Selbstverständlich müssen Sie sorgfältiger vorgehen, wenn Sie auch in die Datei schreiben wollen. Durch zu frühes erneutes Öffnen der Datei nach dem Schreiben kann sich ein inkonsistenter Zustand ergeben, weil `finalize()` und damit `close()` noch nicht ausgeführt wurden. Wenn Sie

den Typ von `InputStream` nicht genau kennen, rufen Sie am besten `close()` selbst auf.

FilterInputStream und FilterReader

Diese »abstrakten« Klassen (in Wirklichkeit ist nur `FilterReader` abstrakt) bieten einen »Durchlauf« für alle Standardmethoden von `InputStream` (oder `Reader`). Sie selbst enthalten einen anderen Datenstream weiter unten in der Filterkette, an die sie alle Methodenaufrufe abgeben. Sie implementieren nichts Neues, gestatten es aber, verschachtelt zu werden:

```
InputStream      s  = getAnInputStreamFromSomewhere();
FilterInputStream s1 = new FilterInputStream(s);
FilterInputStream s2 = new FilterInputStream(s1);
FilterInputStream s3 = new FilterInputStream(s2);

... s3.read() ...
```

Wenn eine Leseoperation auf den gefilterten Datenstream `s3` ausgeführt wird, wird die Anfrage `s2` übergeben. Dann macht `s2` genau das gleiche wie `s1`, und schließlich wird `s` aufgefordert, die Bytes bereitzustellen. Unterklassen von `FilterInputStream` führen eine gewisse Verarbeitung der durchfließenden Bytes aus. Diese im obigen Beispiel eher umständliche »Verkettung« kann eleganter geschrieben werden:

```
s3 = new FilterInputStream(new FilterInputStream(new FilterInputStream(s)));
```

Sie sollten diese Form soweit möglich immer in Ihrem Code verwenden. Sie drückt die Verschachtelung verketteter Filter deutlich aus. Außerdem kann sie leicht analysiert und »laut gelesen« werden, indem man ab dem innersten Datenstream `s` liest, bis man den äußersten Datenstream `s3` erreicht.



`FilterReader` wird als abstrakt deklariert, deshalb können Sie davon keine Instanzen erstellen (was wir im vorherigen Beispiel mit `FilterInputStream` gemacht haben). Außerdem können Sie die zwei Filtertypen nicht beliebig mischen. Bytes und Zeichen lassen sich nicht mischen; `available()` und `ready()` werden nicht korrekt übergeben. Man könnte aber etwa schreiben: `new FilterReaderSubclass(new InputStreamReader(new FilterInputStream(...weitere FilterInputStreams...)))`.

Im nächsten Abschnitt betrachten wir die Subklassen von `FilterInputStream` und die jeweiligen Gegenstücke von `Reader`.

BufferedInputStream und BufferedReader

Das sind zwei der nützlichsten Datenstreams. Sie implementieren die vollen Fähigkeiten der Methoden von `InputStream` und `Reader`, jedoch durch Verwendung eines gepufferten Byte- bzw. Zeichenarrays, der sich als Cache für weitere Leseoperationen verhält. Dadurch werden die gelesenen »Stückchen« von den größeren Blöcken, in denen Datenstreams am effizientesten gelesen werden (z.B. von Peripheriegeräten, Dateien im Dateisystem oder im Netz), abgekoppelt. Ferner ermöglicht es den Datenstreams, Daten voranzulesen.

Da das Puffern von `BufferedInputStream` (`BufferedReader`) so hilfreich ist, und das auch die einzigen Klassen sind, die `mark()` und `reset()` richtig abarbeiten, möchte man sich wünschen, daß jeder Eingabestream (`Reader`) diese wertvollen Fähigkeiten irgendwie nutzt. Normalerweise hat man kein Glück, weil diese Datenstreamklassen sie nicht implementieren. Sie haben aber bereits eine Möglichkeit gesehen, durch die sich Filterstreams um andere Datenstreams »herumwickeln« können. Wenn ein gepufferter `FileInputStream` (`FileReader`) korrekt markieren und zurücksetzen soll, schreiben Sie folgendes:

```
InputStream  s = new BufferedInputStream(new FileInputStream("foo"));
Reader       r = new BufferedReader(new FileReader("foo"));
```

Damit haben Sie einen gepufferten Eingabestream auf der Grundlage der Datei »foo«, die `mark()` und `reset()` unterstützt.



`BufferedReader` ist eigentlich keine Subklasse von `FilterReader`, läßt sich aber wie eine solche »verschachteln«, und sie implementiert alle ihre Methoden völlig analog mit `BufferedInputStream`, so daß die Parallele zwischen den zwei Klassen

greift.

Darüber hinaus hat `BufferedReader` eine spezielle Methode, um eine einzelne Zeile (die mit `'\r'`, `'\n'` oder `'\r\n'` endet) zu lesen:

```
BufferedReader r = new BufferedReader(new FileReader("foo"));
String line = r.readLine(); // Nächste Zeile lesen
```

Jetzt wird die Leistung verschachtelter Datenstreams langsam klar. Jede von einem gefilterten Datenstream bereitgestellte Fähigkeit kann durch Verschachtelung von einem anderen Datenstream genutzt werden. Selbstverständlich ist durch Verschachtelung der Filterstreams jede Kombination dieser Fähigkeiten in jeder beliebigen Reihenfolge möglich.

DataInputStream

Alle Methoden dieser Klasse sind in einer separaten Schnittstelle definiert, die von `DataInputStream` und `RandomAccessFile` (einer weiteren Klasse in `java.io`) implementiert wird. Diese Schnittstelle ist allgemein, so daß Sie sie in Ihren eigenen Klassen benutzen können. Sie heißt `DataInput`.

Die DataInput-Schnittstelle

Wenn Sie häufigen Gebrauch von Datenstreams machen, werden Sie bald feststellen, daß Byte-Streams kein Format bieten, in das alle Daten eingezwängt werden können. Vor allem die primitiven Typen der Java-Sprache können in den bisher behandelten Datenstreams nicht gelesen werden. Die `DataInput`-Schnittstelle spezifiziert Methoden einer höheren Ebene zum Lesen und Schreiben, die komplexere Datenstreams unterstützen. Diese Schnittstelle definiert folgende Methoden:

```
void readFully(byte[] bbuffer) throws IOException;
void readFully(byte[] bbuffer, int offset, int length) throws IOException;
int skipBytes(int n) throws IOException;

boolean readBoolean() throws IOException;
byte readByte() throws IOException;
int readUnsignedByte() throws IOException;
short readShort() throws IOException;
int readUnsignedShort() throws IOException;
char readChar() throws IOException;
int readInt() throws IOException;
long readLong() throws IOException;
float readFloat() throws IOException;
double readDouble() throws IOException;

String readLine() throws IOException;
String readUTF() throws IOException;
```

Die ersten drei Methoden sind lediglich neue Bezeichnungen für `skip()` und die zwei vorher behandelten Formen von `read()`. Die nächsten zehn Methoden lesen einen Primitivtyp bzw. dessen vorzeichenloses Gegenstück (nützlich für die effiziente Verwendung aller Bits in einem Binärstream). Diese Methoden müssen eine Ganzzahl mit einer breiteren Größe ausgeben. Da Ganzzahlen in Java Vorzeichen haben, passen die vorzeichenlosen Werte nicht in kleinere Typen. Die letzten zwei Methoden lesen eine neue Zeile (`'\r'`, `'\n'` oder `'\r\n'`) aus dem Datenstream - beendete Zeichenketten (die erste in ASCII und die zweite in Unicode UTF-8).

Da Sie nun wissen, wie die von `DataInputStream` implementierte Schnittstelle aussieht, betrachten wir sie in Aktion:

```
DataInputStream s = new DataInputStream(getNumericInputStream());
```

```
long size = s.readLong(); // Anzahl Elemente im Datenstream
```

```
while (size-- > 0) {
    if (s.readBoolean()) { // Soll ich dieses Element verarbeiten?
        int anInteger = s.readInt();
        int magicBitFlags = s.readUnsignedShort();
        double aDouble = s.readDouble();
    }
}
```



```

        if ((magicBitFlags & 01000000) != 0) {
            . . .      // Das High-Bit ist gesetzt, etwas Besonderes damit anfangen
        }
        . . .      // Verarbeite anInteger und aDouble
    }
}

```

Die Klasse implementiert eine Schnittstelle für alle ihre Methoden, deshalb können Sie auch folgende Schnittstelle verwenden:

```

DataInput  d = new DataInputStream(new FileInputStream("Irgendwas"));
String     line;

```

```

while ((line = d.readLine()) != null) {
    . . .      // Verarbeite die Zeile
}

```

EOFException

Auf die meisten Methoden von `DataInputStream` trifft folgendes zu: Wird das Ende des Datenstreams erreicht, werfen sie `EOFException` aus. Das ist sehr hilfreich, denn es ermöglicht Ihnen, alle Verwendungen von `-1` in den bisherigen Beispielen besser zu schreiben:

```

DataInputStream  s = new DataInputStream(getAnInputStreamFromSomewhere());

try {
    while (true) {
        byte  b = (byte) s.readByte();
        . . .      // Verarbeite Byte b
    }
} catch (EOFException e) {
    . . .      // Datenstreamende erreicht
}

```

Das funktioniert bei allen `read`-Methoden von `DataInputStream` außer den letzten zwei.



`skipBytes()` bewirkt am Streamende nichts (außer vielleicht ein paar nutzlosen Schleifen), `readLine()` gibt `null` zurück, und `readUTF()` könnte `UTFDataFormatException` auswerfen, falls sie das Problem überhaupt feststellt. (Diese drei Methoden sind nicht die besten Beispiele für gut geschriebenen Java-Code.)

PushbackInputStream und PushbackReader

Die Filterstreamklasse `PushbackInputStream` (bzw. `PushbackReader`) wird üblicherweise in Parsern benutzt, um ein einzelnes Byte (Zeichen) der Eingabe (nach dem Lesen) »zurückzuschieben«, während versucht wird, die nächste Aktion zu ermitteln. Das ist eine vereinfachte Version von `mark()` und `reset()`. Sie erweitert die `InputStream`-Standardmethoden um `unread()`. Wie Sie sich denken können, gibt diese Methode vor, das in ihr durchgereichte Byte (Zeichen) nie gelesen zu haben. Dann übergibt sie dieses Byte (Zeichen) dem nächsten `read()` als Ausgabewert. In Version 1.1 sind neue Methoden zum »Zurücklesen« (`unread()`) eines ganzen Puffers und eines Teilbereichs. Das bedeutet, daß es jetzt drei Formen von `unread()` gibt, die den drei Standardformen von `read()` entsprechen.

Das folgende Beispiel ist eine einfache Implementierung von `readLine()` anhand dieser Klasse (eine Anpassung der Implementierung aus `DataInputStream.java`):

```

public class SimpleLineReader {
    private FilterInputStream  s;

    public SimpleLineReader(InputStream  anIS) {

```

```

        s = new DataInputStream(anIS);
    }

    . . .    // Weitere read()-Methoden mit Datenstream s

    public String  readLine() throws IOException {
        char[]  buffer = new char[100];
        int      offset = 0;
        byte     thisByte;

        try {
loop:      while (offset < buffer.length) {
                switch (thisByte = (byte) s.read()) {
                    case '\n':
                        break loop;
                    case '\r':
                        byte  nextByte = (byte) s.read();

                        if (nextByte != '\n') {
                            if (!(s instanceof PushbackInputStream)) {
                                s = new PushbackInputStream(s);
                            }
                            ((PushbackInputStream) s).unread(nextByte);
                        }
                        break loop;
                    default:
                        buffer[offset++] = (char) thisByte;
                        break;
                }
            }
        } catch (EOFException e) {
            if (offset == 0)
                return null;
        }
        return String.copyValueOf(buffer, 0, offset);
    }
}

```

Das zeigt verschiedene Dinge auf. In diesem Beispiel ist `readLine()` auf das Lesen der ersten 100 Zeichen der Zeilen begrenzt. (In dieser Hinsicht wird aufgezeigt, wie eine allgemeine Zeilenverarbeitung *nicht* geschrieben werden sollte - wir wollen ja Zeilen jeder Größe lesen.) Außerdem werden wir daran erinnert, daß wir mit `break` aus einer äußeren Schleife ausbrechen können und wie eine Zeichenkette (string) aus einem Zeichenarray erzeugt wird (in diesem Fall aus einer »Scheibe« des Zeichenarrays). In diesem Beispiel wird auch die Standardverwendung von `read()` in `InputStream` zum Lesen der einzelnen Bytes aufgezeigt. Das Stream-Ende wird durch Einbinden in `DataInputStream` und `catch EOFException` festgelegt.

Ein ungewöhnlicher Aspekt ist bei diesem Beispiel die Art, wie `PushbackInputStream` verwendet wird. Um sicher zu sein, daß `'\n'` nach `'\r'` ignoriert wird, muß ein Zeichen vorausgelesen werden. Ist dieses Zeichen kein `'\n'`, muß es zurückgeschoben werden. Wir betrachten die Quellzeilen, beginnend mit `if (...instanceof...)`, als ob wir nichts über den Datenstream `s` wüßten. Die allgemein angewandte Technik ist lehrreich. Erstens sehen wir, ob `s` bereits eine Instanz (`instanceof`) der einen oder anderen Art von `PushbackInputStream` ist. Trifft das zu, können wir ihn direkt verwenden. Andernfalls wird der aktuelle Datenstream (egal welcher) in einen neuen `PushbackInputStream` gesetzt, und dieser neue Datenstream wird verwendet.

Die nächste Zeile möchte die Methode `unread()` aufrufen. Das Problem dabei ist, daß `s` den *Kompilierzeit*typ `FilterInputStream` hat, den somit diese Methode nicht versteht. Die zwei vorherigen Zeilen gewährleisteten jedoch, daß `PushbackInputStream` der *Laufzeit*typ des Datenstreams in `s` ist, so daß Sie ihn problemlos in diesen Typ umwandeln und `unread()` aufrufen können.



Dieses Beispiel ist aus Demonstrationszwecken etwas ungewöhnlich ausgefallen. Sie könnten auch eine `PushbackInputStream`-Variable deklarieren und darin `DataInputStream` einbinden. Umgekehrt könnte der Konstruktor von `SimpleLineReader` prüfen, ob sein Argument bereits von der richtigen Klasse ist, wie `PushbackInputStream` das macht, bevor ein neuer `DataInputStream` erstellt wird. Interessant an diesem Ansatz ist das Einbinden einer Klasse bei Bedarf. Das ist bei jedem `InputStream` möglich und erfordert keinen zusätzlichen Aufwand. Beide Ansätze gelten als gute Designprinzipien.

Bisher wurden noch nicht alle Subklassen von `FilterInputStream` beschrieben. Nun ist es an der Zeit, zu den direkten Unterklassen von `InputStream` zurückzukehren.

ObjectInputStream

Nachdem Sie ein komplexes Geflecht aus untereinander verbundenen Objekten erstellt haben, ist oft die Möglichkeit nützlich, den Zustand all dieser Objekte gleichzeitig »speichern« zu können. Das vereinfacht das Kopieren zu Zwecken wie Backup oder Rückgängigmachen und Wiederherstellen. Außerdem bleiben die Objekte im Dateisystem erhalten und können später wieder »zum Leben erweckt« werden. Darüber hinaus können Objekte im Internet gemeinsam »auf die Reise gehen« und sicher am anderen Ende ankommen. (Sie können bereits ganze Klassen auf diese Weise senden und somit neue Instanzen am anderen Ende erstellen. Möchten Sie aber den *Inhalt* eines lokalen Objekts übertragen, brauchen Sie etwas Neues.)

Das JDK 1.1 führte das Konzept der Serialisation ein. Das bedeutet im wesentlichen, daß ein Objekt leicht und sicher in einen Datenstream und wieder zurück verwandelt werden kann. In Verbindung mit seiner »Bruderklasse« `ObjectOutputStream` bewirkt `ObjectInputStream` genau das.



Seit JDK 1.2 beta 3 haben Output Streams defaultmäßig ein neues Format, das nicht kompatibel mit dem alten ist. Um die Rückwärtskompatibilität zu gewährleisten gibt es die neue Funktion

```
java.io.ObjectOutputStream.useProtocolVersion( ). Verwenden Sie für das alte Format
java.io.ObjectStreamConstants.PROTOCOL_VERSION_1, für das neue Format
java.io.ObjectStreamConstants.PROTOCOL_VERSION_2.
```

Die Input Streams erkennen und benutzen automatisch das richtige Format.

Aus Sicherheitsgründen werden zur Serialisation nur Objekte zugelassen, die zum Austausch zwischen Systemen als »sicher« deklariert wurden. Solche Objekte sind Instanzen von Klassen, die die neue Schnittstelle `Serializable` implementieren. Die meisten internen Systemklassen implementieren `Serializable` nicht, wohl aber viele der mehr »informativen« Klassen.

Alle Methoden, die Instanzen dieser Klasse verstehen, sind in der getrennten Schnittstelle `ObjectInput` definiert, die `ObjectInputStream` implementiert.



Außerdem gibt es eine Schnittstelle namens `Externalizable`, die von `Serializable` abgeleitet ist. Sie gibt Objekten mehr Kontrolle darüber, wie sie geschrieben und gelesen werden. Diese Einrichtungen der unteren Ebene braucht man aber fast nie.

Die ObjectInput-Schnittstelle

Die Schnittstelle `ObjectInput` leitet die Schnittstelle `DataInput` ab, wobei sie alle ihre Methoden erbt und darüber hinaus eine neue Methode der oberen Ebene bereitstellt, die einen komplexen Typenstream serialisierter Objektdaten unterstützt:

```
Object readObject() throws ClassNotFoundException, IOException;
```

Im folgenden einfachen Beispiel wird ein solcher Datenstream gelesen, der im »Bruderbeispiel« (ObjectOutputStream) in einem späteren Beispiel der heutigen Lektion produziert wird:

```
FileInputStream    s    = new FileInputStream("objectFileName");
ObjectInputStream  ois = new ObjectInputStream(s);

int      i      = ois.readInt();           // Benutzt die DataInput-Methode
String   today  = (String) ois.readObject();
Date     date   = (Date)   ois.readObject();
s.close();
```



Denken Sie daran, daß Sie die Ergebnisse von readObject() vor Verwendung in die erwartete Klasse immer konvertieren müssen. Sogar Arrays werden als Objekte gesendet und müssen vor Verwendung in die richtigen Kompilierzeittypen konvertiert werden.



Daneben gibt es viele weitere nützliche Anwendungen der Serialisation (beispielsweise können Klassen im Datenstream enthalten sein, sich automatisch Versionen zuweisen usw.). Lesen Sie die Kommentare zu diesen Klassen oder die Dokumentation des JDK 1.2.

PipedInputStream und PipedReader

Diese Klassen und ihre Schwestern PipedOutputStream und PipedReader werden später in der heutigen Lektion behandelt. Vorläufig genügt zu wissen, daß sie zusammen eine einfache zweiwegige Kommunikation zwischen Threads ermöglichen.

SequenceInputStream

Soll aus zwei Datenstreams ein zusammengesetzter Datenstream gebildet werden, wird SequenceInputStream verwendet:

```
InputStream  s1 = new FileInputStream("theFirstPart");
InputStream  s2 = new FileInputStream("theRest");

InputStream  s  = new SequenceInputStream(s1, s2);

... s.read() ... // Liest nacheinander aus jedem Datenstream
```

Wir hätten das gleiche Ergebnis durch abwechselndes Lesen der Dateien auch »simulieren« können. Was aber, wenn wir den zusammengesetzten Datenstream s einer anderen Methode übergeben wollen, die nur einen InputStream erwartet? Hier ein Beispiel (mit s), bei dem die Zeilen der zwei vorherigen Dateien durch ein übliches Nummerierungsschema nummeriert werden:

```
LineNumberInputStream  aLNIS = new LineNumberInputStream(s);

... aLNIS.getLineNumber() ...
```



Diese Art der Verkettung von Datenstreams ist besonders nützlich, wenn Länge und Herkunft der Datenstreams nicht bekannt sind.

Wenn Sie mehr als zwei Datenstreams verketteten wollen, versuchen Sie es so:

```
Vector  v = new Vector();
...    // Setze alle Datenstreams und füge jeden einzelnen zum Vektor hinzu
InputStream  s1 = new SequenceInputStream(v.elementAt(0), v.elementAt(1));
```

```
InputStream s2 = new SequenceInputStream(s1, v.elementAt(2));
InputStream s3 = new SequenceInputStream(s2, v.elementAt(3));
. . .
```



Ein *Vektor* (ein Objekt der Klasse `Vector`) ist ein Objektarray, das dynamisch seine Größe verändern kann, dem Elemente hinzugefügt werden können, das mit `elementAt()` einzelne Elemente ansprechen und dessen Inhalt aufgelistet werden kann.

Viel einfacher ist aber die Verwendung eines anderen Konstruktors, den `SequenceInputStream` bietet:

```
InputStream s = new SequenceInputStream(v.elements());
```

Hierfür ist eine Aufzählung aller Datenstreams erforderlich, die kombiniert werden sollen. Im Anschluß wird ein einzelner Datenstream zurückgegeben, der die Daten nacheinander liest.

StringBufferInputStream und StringReader

`StringBufferInputStream` (`StringReader`) ist genau wie `ByteArrayInputStream` (`CharArrayReader`), basiert aber nicht auf einem `Byte-` bzw. `Zeichenarray`, sondern auf einem `String`:

```
String      buffer = "Now is the time for all good men to come...";
InputStream s      = new StringBufferInputStream(buffer);
Reader      r      = new StringReader(buffer);
```

Alle Kommentare zu `ByteArrayInputStream` (`CharArrayReader`) treffen auch hier zu (siehe ersten Abschnitt über diese Klassen).



Die Bezeichnung `StringBufferInputStream` ist nicht gut gelungen, weil dieser Eingabestream eigentlich auf einem `String` basiert. `StringInputStream` wäre besser geeignet. Außerdem handhabt er `reset()` durch Zurücksetzen der Zeichenkette an den Anfang, und `markSupported()` gibt `false` zurück (ist also nicht ganz symmetrisch mit `StringReader`). Das sind im wesentlichen Bugs, die aus Version 1.0 stammen.

Ausgabedatenstreams und Writer

Ausgabedatenstreams und `Writer` werden fast ausnahmslos mit einem brüderlichen `InputStream` (bzw. `Reader`) gepaart. Führt ein `InputStream` (`Reader`) eine bestimmte Operation aus, wird die umgekehrte Operation vom `OutputStream` (`Writer`) ausgeführt. Was das bedeuten soll, sehen Sie in Kürze.

Die abstrakten Klassen OutputStream und Writer

`OutputStream` ist die abstrakte Klasse, die die grundlegenden Arten definiert, in der eine Quelle (Erzeuger) einen `Bytestream` in ein Ziel schreiben kann. Die Identität des Ziels und die Art der Beförderung und Speicherung der Bytes sind nicht relevant. Bei der Verwendung eines Ausgabestreams sind *Sie* die Quelle der Bytes. Das ist alles, was Sie wissen müssen.

`Writer` ist eine abstrakte Klasse, die die grundlegenden Arten definiert, in der eine Quelle (Erzeuger) einen `Bytestream` in ein Ziel schreiben kann. Sie und alle ihre Unterklassen entsprechen `OutputStream` und ihren Unterklassen, ausgenommen, daß sie als Grundeinheiten nicht Bytes, sondern Zeichen verwenden.

write()

Die wichtigste Methode für den Erzeuger eines Ausgabestreams (oder `Writers`) ist diejenige, die Bytes (Zeichen) in das Ziel schreibt. Diese Methode ist `write()`, die es in verschiedenen Varianten gibt, wie Sie in den folgenden Beispielen sehen werden.



Alle Varianten der `write()`-Methode müssen warten, bis die gesamte angeforderte Ausgabe geschrieben ist. Diese Einschränkung soll Sie aber nicht beunruhigen. Wenn Sie sich nicht erinnern, warum das so ist, lesen Sie den Hinweis zu `read()` unter `InputStream`.

```
OutputStream s      = getAnOutputStreamFromSomewhere();
Writer w          = getAWriterFromSomewhere();
byte[] bbuffer = new byte[1024];           // Größe kann beliebig sein
char[] cbuffer = new char[1024];

fillInData(bbuffer);    fillInData(cbuffer);    // Die Daten, die wir ausgeben
                                                    // wollen

s.write(bbuffer);
w.write(cbuffer);
```

Sie können auch ein »Scheibchen« Ihres Puffers schreiben, indem Sie den Versatz und die gewünschte Länge als Argumente für `write()` angeben:

```
s.write(bbuffer, 100, 300);
w.write(cbuffer, 100, 300);
```

Dadurch werden die Bytes (Zeichen) 100 bis 399 ausgegeben. Ansonsten ist das Verhalten genauso wie bei der vorherigen `write()`-Methode. Im derzeitigen Release benutzt die Standardimplementierung der ersten Form von `write()` die zweite Alternative:

```
public void write(byte[] b) throws IOException {    /* Von OutputStream.java */
    write(b, 0, b.length);
}
public void write(char[] cbuf) throws IOException {    /* Von Writer.java */
    write(cbuf, 0, cbuf.length);
}
```

Letztlich können Sie Bytes einzeln ausgeben:

```
while (thereAreMoreBytesToOutput() && thereAreMoreCharsToOutput()) {
    byte b = getNextByteForOutput();
    char c = getNextCharForOutput();

    s.write(b);
    w.write(c);
}
```



`Writer` hat eigentlich zwei zusätzliche Methoden zum Schreiben einer Zeichenkette (`string`) und einer Zelle einer Zeichenkette. Sie werden genauso benutzt wie die ersten zwei Formen von `write()` (lediglich `cbuffer` wird durch `string` ersetzt).

flush()

Da wir nicht wissen, womit ein Ausgabestream (`Writer`) verbunden ist, können wir mit `flush()` die Leerung der Ausgabe durch einen gepufferten Cache anfordern, um sie (zeitgerecht oder überhaupt) zu erhalten. Die `OutputStream`-Version dieser Methode bewirkt nichts. Von ihr wird lediglich erwartet, diese Version durch Subklassen, die `flush()` voraussetzen (z.B. `BufferedOutputStream` und `PrintStream`), mit nichttrivialen Aktionen zu überschreiben.

close()

Wie bei `InputStream` (oder `Reader`) sollte ein `OutputStream` normalerweise explizit geschlossen werden, damit die von

ihm beanspruchten Ressourcen freigegeben werden. Im übrigen trifft alles zu, was über `close()` in Zusammenhang mit `InputStream` gesagt wurde.

Alle Ausgabestreams stammen von der abstrakten Klasse `OutputStream`, und alle `Writer` stammen von `Writer` ab und haben die oben beschriebenen Methoden.



Konkrete Subklassen von `OutputStream` brauchen nur die Form von `write()` ohne Argumente implementieren, um alle übrigen Methoden zum Arbeiten zu bringen (`OutputStream` hat in der Standardimplementierung die Methoden `close()` und `flush()`, die nichts bewirken). Subklassen von `Writer` müssen aber sowohl `close()` als auch die Form von `write()` mit den drei Argumenten implementieren.

ByteArrayOutputStream und CharArrayWriter

Das Gegenstück von `ByteArrayInputStream` (`CharArrayReader`), das einen Eingabestream für ein Byte- bzw. Zeichenarray erzeugt, ist `ByteArrayOutputStream` (`CharArrayWriter`), der einen Ausgabestream an ein Byte- oder Zeichenarray übergibt:

```
OutputStream s = new ByteArrayOutputStream();
Writer       w = new CharArrayWriter();
s.write(123);
w.write('\n');
. . .
```

Die Größe eines internen Byte- bzw. Zeichenarrays wächst nach Bedarf, um einen Datenstream jeder beliebigen Länge zu speichern. Sie können auf Wunsch eine Anfangskapazität als Hilfe für die Klasse festlegen:

```
OutputStream s = new ByteArrayOutputStream(1024 * 1024); // 1 Megabyte
Writer       w = new CharArrayWriter(1024 * 1024);
```



Damit haben Sie die ersten Beispiele des Erstellens von Ausgabestreams (und Writern) gesehen. Diese neuen Datenstreams werden an die einfachsten aller möglichen Datenquellen angehängt - ein Byte- bzw. Zeichenarray im Speicher des lokalen Rechners.

Nachdem `ByteArrayOutputStream` `s` (bzw. `CharArrayWriter` `w`) gefüllt wurde, kann er Daten an einen anderen Ausgabestream (oder Schreiber) ausgeben:

```
OutputStream      anotherOutputStream = getTheOtherOutputStream();
Writer            anotherWriter       = getTheOtherWriter();
ByteArrayOutputStream s = new ByteArrayOutputStream();
CharArrayWriter    w = new CharArrayWriter();
```

```
fillWithUsefulData(s);    fillWithUsefulData(w);
s.writeTo(anotherOutputStream);
w.writeTo(anotherWriter);
```

Außerdem kann er als Byte- oder Zeichenarray herausgezogen oder in eine Zeichenkette (string) konvertiert werden:

```
byte[]  bbuffer      = s.toByteArray();
char[]  cbuffer      = w.toCharArray();
String  streamString  = s.toString();
String  writerString  = w.toString();
String  streamUnicodeString = s.toString(upperByteValue);
```



Die letzte Methode ermöglicht das »Simulieren« von Unicode-Zeichen (16 Bit) durch Auffüllen der niedrigen Bytes mit ASCII und Spezifizieren eines oberen Byte (normalerweise 0), um eine Unicode-Zeichenkette zu erzeugen.

ByteArrayOutputStream (und CharArrayWriter) haben zwei Utility-Methoden: Eine gibt die aktuelle Anzahl der im internen Byte- bzw. Zeichenarray gespeicherten Bytes (Zeichen) aus, die andere setzt das Array zurück, so daß der Datenstream von Anfang an erneut geschrieben werden kann:

```
int  sizeofMyByteArray = s.size();
int  sizeofMyCharArray = w.size();

s.reset();      // s.size() würde jetzt 0 zurückgeben
w.reset();      // w.size() würde jetzt 0 zurückgeben
s.write(123);
w.write('\n');
. . .
```



Writer hat einen Verwandten namens StringWriter, der ein wenig aus der Reihe tanzt - er hat kein Reader-Gegenstück und ist fast identisch mit CharArrayWriter (er fügt getBuffer() hinzu, was String zurückgibt, und implementiert writeTo(), toCharArray() und reset() nicht). Die Darstellung und Benutzung von Zeichenketten und Zeichenarrays sind sehr ähnlich, so daß diese Klasse überflüssig ist.

FileOutputStream und FileWriter

Eine der häufigsten Verwendungen von Datenstreams und historisch die älteste ist das Anhängen von Datenstreams an Dateien im Dateisystem. Hier wird beispielsweise ein solcher Ausgabestream (oder Writer) auf einem Unix-System erstellt:

```
OutputStream  s = new FileOutputStream("/Irgendein/Pfad/und/Dateiname");
Writer        w = new FileWriter("/Irgendein/Pfad/und/Dateiname.utf8");
```



Applets, die versuchen, solche Datenstreams im Dateisystem zu öffnen, zu lesen oder zu schreiben, können Sicherheitsverletzungen verursachen. Weitere Einzelheiten finden Sie im Hinweis unter FileInputStream und FileReader.



FileOutputStream (jedoch nicht FileWriter) hat auch einen Konstruktor, der einen String und einen booleschen Wert annimmt, um zu entscheiden, ob die Daten an die Datei angehängt werden müssen.

Sie können Datenstreams auch aus einem zuvor aktivierten FileDescriptor oder einer Datei erstellen:

```
OutputStream  s = new FileOutputStream(FileDescriptor.out); /* Standardeingabe */
Writer        w = new FileWriter(FileDescriptor.err);      /* Standardfehler */
OutputStream  s = new FileOutputStream(new File("/Irgendein/Pfad/und/Dateiname"));
Writer        w = new FileWriter(new File("/Irgendein/Pfad/und/Dateiname.utf8"));
```

FileWriter ist eigentlich eine triviale Subklasse der weiteren writer-Klasse OutputStreamWriter, die jeden beliebigen OutputStream kapseln und ihn in einem Writer vom Typ char umwandeln kann. Somit besteht die Implementierung von FileWriter lediglich aus der Aufforderung von OutputStreamWriter, (selbst) einen FileOutputStream zu kapseln:

```
FileOutputStream  aFOS = new FileOutputStream("aFileName");
```

```
FileDescriptor  myFD = aFOS.getFD();
```

```
/* aFOS.finalize(); */ // Aktiviere close() bei automatischem Aufruf des Garbage-
```

//Collectors

Ein Aspekt ist ganz klar: `getFD()` gibt die Dateisignatur (`FileDescriptor`) zurück, auf der der Datenstream basiert. Der zweite Aspekt ist ein Kommentar, der Sie daran erinnern soll, daß Sie sich um das Schließen dieses Datenstreamtyps keine Gedanken machen müssen. Die Implementierung von `finalize()` besorgt das automatisch. (Siehe Erläuterungen unter `FileInputStream` und `FileReader`.)

FilterOutputStream und FilterWriter

Diese »abstrakten« Klassen (in Wirklichkeit ist nur `FilterWriter` abstrakt) bieten einen »Durchlauf« für alle Standardmethoden von `OutputStream` (oder `Writer`). Sie selbst enthalten einen anderen Datenstream weiter unten in der Filterkette, an die sie alle Methodenaufrufe abgeben. Sie implementieren nichts Neues, gestatten aber ihr eigenes Verschachteln:

```
OutputStream      s  = getAnOutputStreamFromSomewhere();
FilterOutputStream s1 = new FilterOutputStream(s);
FilterOutputStream s2 = new FilterOutputStream(s1);
FilterOutputStream s3 = new FilterOutputStream(s2);

... s3.write(123) ...
```

Wenn eine Schreiboperation auf den gefilterten Datenstream `s3` ausgeführt wird, wird die Anfrage `s2` übergeben. Dann macht `s2` genau das gleiche wie `s1`, und schließlich wird `s` aufgefordert, die Bytes bereitzustellen. Subklassen von `FilterOutputStream` führen eine gewisse Verarbeitung der durchfließenden Bytes aus. Diese im obigen Beispiel eher umständliche »Verkettung« kann eleganter geschrieben werden. Wie das gemacht wird, finden Sie unter der »Bruderklasse« `FilterInputStream`.

Im nächsten Abschnitt betrachten wir die Subklassen von `FilterOutputStream`.

BufferedOutputStream und BufferedWriter

Das sind zwei der nützlichsten Datenstreams. Sie implementieren die vollen Fähigkeiten der Methoden von `OutputStream` und `Writer`, jedoch durch Verwendung eines gepufferten Byte- bzw. Zeichenarrays, der sich als Cache für weitere Schreiboperationen verhält. Dadurch werden die geschriebenen »Stückchen« von den größeren Blöcken, in denen Datenstreams am effizientesten geschrieben werden (z.B. in Peripheriegeräten, Dateien im Dateisystem oder im Netz), abgekoppelt.

`BufferedOutputStream` (`BufferedWriter`) ist eine der wenigen Klassen der Java-Bibliothek, die eine nichttriviale Version von `flush()` implementieren. Sie bewirkt, daß die geschriebenen Bytes (Zeichen) durch den Puffer geschoben und auf der anderen Seite ausgegeben werden. Da das Puffern von `BufferedOutputStream` (`BufferedWriter`) so hilfreich ist, möchte man sich wünschen, daß jeder Ausgabestream (`Writer`) diese wertvollen Fähigkeiten irgendwie nutzt. Zum Glück können Sie jeden Ausgabestream (oder `Writer`) umgehen, um genau das zu erreichen:

```
OutputStream  s = new BufferedOutputStream(new FileOutputStream("foo"));
Writer        w = new BufferedWriter (new FileWriter("foo.utf8"));
```

Damit haben Sie einen gepufferten Ausgabestream (`Writer`) auf der Grundlage der Datei »foo«, die `flush()` unterstützt.



`BufferedWriter` ist eigentlich keine Subklasse von `FilterWriter`, läßt sich aber wie eine solche »verschachteln«. Außerdem hat sie die einzigartige Methode `newLine()`, die Zeichen für neue Zeilenanfänge passend zu dem lokalen System setzt, auf dem Java läuft.

Jede von einem gefilterten Ausgabedatenstream (oder `Writer`) bereitgestellte Fähigkeit kann durch Verschachtelung von einem anderen Datenstream genutzt werden. Selbstverständlich ist durch Verschachtelung der Filterstreams jede Kombination dieser Fähigkeiten in jeder beliebigen Reihenfolge möglich.

DataOutputStream

Alle Methoden dieser Klasse sind in einer separaten Schnittstelle definiert, die von `DataOutputStream` und

RandomAccessFile implementiert wird. Diese Schnittstelle ist allgemein, so daß Sie sie in Ihren eigenen Klassen benutzen können. Sie heißt DataOutput .

Die DataOutput-Schnittstelle

In Zusammenhang mit dem Gegenstück DataInput bietet DataOutput Methoden höherer Ebene zum Lesen und Schreiben von Daten. Anstatt sich mit Bytes zu befassen, schreibt diese Schnittstelle die primitiven Typen von Java direkt:

```
void write(int i) throws IOException;
void write(byte[] buffer) throws IOException;
void write(byte[] buffer, int offset, int length) throws IOException;

void writeBoolean(boolean b) throws IOException;
void writeByte(int i) throws IOException;
void writeShort(int i) throws IOException;
void writeChar(int i) throws IOException;
void writeInt(int i) throws IOException;
void writeLong(long l) throws IOException;
void writeFloat(float f) throws IOException;
void writeDouble(double d) throws IOException;

void writeBytes(String s) throws IOException;
void writeChars(String s) throws IOException;
void writeUTF(String s) throws IOException;
```

Zu den meisten dieser Methoden gibt es DataInput-Gegenstücke.

Die ersten drei Methoden spiegeln lediglich die drei Formen von write() wider, die Sie bereits kennengelernt haben. Die nächsten acht Methoden schreiben jeweils einen primitiven Typ. Die letzten drei Methoden schreiben eine aus Bytes oder Zeichen bestehende Zeichenkette in den Datenstream: die erste als 8-Bit-Bytes, die zweite als 16- Bit-Zeichen im binären Unicode und die dritte als speziellen Unicode-Datenstream (UTF-8) (der von readUTF() in DataInput gelesen werden kann).



Die Lesemethoden für vorzeichenlose Datentypen von DataInput haben keine DataOutput -Gegenstücke. Sie können die erforderlichen Daten über die Vorzeichenmethoden von DataOutput ausgeben, weil sie int-Argumente akzeptieren und auch die richtige Anzahl Bits für die vorzeichenlose Ganzzahl einer bestimmten Größe schreiben. Die Methode, die diese Ganzzahl liest, muß das Vorzeichenbit richtig interpretieren.

Da Sie nun wissen, wie die von DataOutputStream implementierte Schnittstelle aussieht, betrachten wir sie in Aktion:

```
DataOutputStream s = new DataOutputStream(getNumericOutputStream());
long size = getNumberOfItemsInNumericStream();

s.writeLong(size);

for (int i = 0; i < size; ++i) {
    if (shouldProcessNumber(i)) {
        s.writeBoolean(true); // Sollte dieses Element verarbeiten
        s.writeInt(theIntegerForItemNumber(i));
        s.writeShort(theMagicBitFlagsForItemNumber(i));
        s.writeDouble(theDoubleForItemNumber(i));
    } else
        s.writeBoolean(false);
}
```

Das ist das genaue Gegenstück des mit DataInput aufgeführten Beispiels. Zusammen bilden sie ein Paar, das ein bestimmtes strukturiertes Primitivtypen-Array über jeden Datenstream (bzw. die Transportschicht) austauschen kann. Verwenden Sie dieses Paar als Sprungbrett für ähnliche Aktionen.

Zusätzlich zur obigen Schnittstelle implementiert die Klasse eine (selbsterklärende) Utility-Methode:

```
int theNumberOfBytesWrittenSoFar = s.size();
```

Verarbeiten einer Datei

Zu den häufigsten Ein- und Ausgabeoperationen zählen das Öffnen einer Datei, das zeilenweise Lesen und Verarbeiten und das Ausgeben dieser Daten in eine andere Datei. Das folgende Beispiel ist ein Prototyp dessen, wie dies in Java realisiert wird:

```
DataInput  aDI = new DataInputStream(new FileInputStream("source"));
DataOutput aDO = new DataOutputStream(new FileOutputStream("dest"));
String      line;

while ((line = aDI.readLine()) != null) {
    StringBuffer modifiedLine = new StringBuffer(line);

    . . .          // Verarbeite modifiedLine
    aDO.writeBytes(modifiedLine.toString());
}
aDI.close();
aDO.close();
```

Möchten Sie das byteweise verarbeiten, schreiben Sie folgendes:

```
try {
    while (true) {
        byte b = (byte) aDI.readByte();
        . . .          // Verarbeite b
        aDO.writeByte(b);
    }
} finally {
    aDI.close();
    aDO.close();
}
```

Der folgende nette Zweizeiler kopiert die Datei:

```
try { while (true) aDO.writeByte(aDI.readByte()); }
finally { aDI.close(); aDO.close(); }
```



Bei zahlreichen Beispielen der heutigen Lektion (darunter die letzten zwei) wird davon ausgegangen, daß sie in einer Methode erscheinen, die IOException in ihrer throws- Klausel hat. Deshalb müssen Sie sich nicht um das Auffangen (catch) dieser Ausnahmen und deren angemessene Handhabung kümmern. Für die Praxis sollte Ihr Code weniger großzügig sein.

PrintStream und PrintReader

Ohne sich möglicherweise dessen bewußt zu sein, sind Sie bereits mit den zwei Methoden der PrintStream-Klasse vertraut. Wenn Sie die Methodenaufrufe

```
System.out.print(. . .)
System.out.println(. . .)
```

verwenden, benutzen Sie eigentlich eine Instanz von PrintStream, die sich in der Variablen out der Klasse System befindet, um die Ausgabe auszuführen. System.err gehört ebenfalls zu PrintStream, und System.in ist ein InputStream.



Auf Unix-Systemen werden diese drei Datenstreams an Standardausgabe, Standardfehler und Standardeingabe angehängt.

PrintStream ist ein Ausgabestream ohne brüderliches Gegenstück. PrintWriter ist einer von nur zwei Writern mit der gleichen Eigenschaft. Da sie normalerweise mit einer Bildschirmausgabe zusammenhängen, implementieren sie flush(). Ferner bieten sie die bekannten Methoden close() und write() sowie eine Fülle von Möglichkeiten zur Ausgabe der primitiven Typen und Zeichenketten von Java:

```
public void write(int byteOrChar); // byte (PrintStream), char (PrintWriter)
public void write(byte[] buffer, int offset, int length); // PrintStream
public void write(char[] buffer, int offset, int length); // PrintWriter
public void write(String string); // Die nächsten zwei Methoden nur in
                                   // PrintWriter
public void write(String string, int offset, int length); // PrintWriter
public void flush(); // (Alles ab hier ist in beiden Klassen)
public void close();

public void print(Object o);
public void print(String s);
public void print(char[] buffer);
public void print(char c);
public void print(int i);
public void print(long l);
public void print(float f);
public void print(double d);
public void print(boolean b);

public void println(Object o);
public void println(String s);
public void println(char[] buffer);
public void println(char c);
public void println(int i);
public void println(long l);
public void println(float f);
public void println(double d);
public void println(boolean b);

public void println(); // Leerzeile ausgeben
```

PrintStream (PrintWriter) kann auch benutzt werden, um einen Ausgabestream zu umwickeln wie eine Filterklasse (trotz der Tatsache, daß PrintWriter keine Subklasse von FilterWriter ist, läßt sie sich wie eine verschachteln):

```
PrintStream s = new PrintStream(new FileOutputStream("foo"));
PrintWriter w = new PrintWriter(new FileWriter("foo.utf8"));

s.println("Das ist die erste Textzeile der Datei foo.");
w.println("Das ist die erste Textzeile der Datei foo.utf8.");
```

Ein zweites Argument für den Konstruktor von PrintStream (oder PrintWriter) ist boolesch und bestimmt, ob der Datenstream automatisch »flushen« soll. Im Fall von true (wahr) wird nach jedem Zeichen, das eine neue Zeile setzt ('\n'), ein flush() gesendet. Bei der Form von write() mit drei Argumenten wird nach jeder Zeichengruppe ein flush() gesendet. PrintWriter handhabt das automatische Flush ein wenig anders - flush() wird nur nach dem Aufruf einer der Methoden println(...) gesetzt.

Das folgende kleine Programm arbeitet wie der Unix-Befehl cat. Es nimmt die Standardeingabe zeilenweise entgegen und gibt sie auf der Standardausgabe aus:

```
import java.io.*; // Das schreiben wir heute nur hier

public class Cat {
    public static void main(String argv[]) {
        DataInput d = new DataInputStream(System.in);
        String line;
```

```

    try { while ((line = d.readLine()) != null)
        System.out.println(line);
    } catch (IOException ignored) { }
}

```

Damit wurden nun alle Subklassen von `FilterOutputStream` beschrieben. Wir wenden uns jetzt den direkten Subklassen von `OutputStream` zu.

ObjectOutputStream

Diese und die Bruderklasse `ObjectInputStream` unterstützen die Serialisation von Objekten (weitere Einzelheiten hierzu finden Sie unter `ObjectInputStream`). Alle Methoden, die Instanzen dieser Klasse verstehen, sind in der getrennten Schnittstelle `ObjectOutput` definiert, die `ObjectOutputStream` implementiert.

Die ObjectOutput-Schnittstelle

Diese Schnittstelle ist von der Schnittstelle `DataOutput` abgeleitet, wobei sie alle ihre Methoden erbt und darüber hinaus eine neue Methode der oberen Ebene bereitstellt, die einen komplexen Typenstream serialisierter Objektdaten unterstützt:

```
void writeObject(Object obj) throws IOException;
```

Im folgenden einfachen Beispiel wird ein Datenstream geschrieben, der im »Bruderbeispiel« (`ObjectInputStream`) in einem früheren Beispiel der heutigen Lektion gelesen wurde:

```

FileOutputStream      s    = new FileOutputStream("objectFileName");
ObjectOutputStream    oos = new ObjectOutputStream(s);

oos.writeInt(12345);           // Benutzt die DataOutput-Methode
oos.writeObject("Today");
oos.writeObject(new Date());
oos.flush();
s.close();

```

PipedOutputStream und PipedWriter

Diese und die Klassen `PipedInputStream` (`PipedReader`) bilden zusammen die Paare, die eine Unix-artige Pipe-Verbindung zwischen zwei Threads herstellen und sorgfältig die gesamte Synchronisation implementieren, die eine sichere Operation dieser Art von gemeinsamer Warteschlange ermöglicht. Die Verbindung wird so eingerichtet:

```

PipedInputStream  sIn  = new PipedInputStream();
PipedOutputStream sOut = new PipedOutputStream(sIn);
PipedReader      wIn  = new PipedReader();
PipedWriter      wOut = new PipedWriter(wIn);

```

Ein Thread schreibt `sOut` (`wOut`), und der andere liest von `sIn` (`wIn`). Durch Einrichten solcher Paare können die Threads in beiden Richtungen problemlos kommunizieren.



`PipedOutputStream` implementiert die zwei Formen von `write()` - eine ohne und eine mit drei Argumenten, während `PipeWriter` nur die Form mit drei Argumenten hat.

Zusammenhängende Klassen

Die übrigen Klassen und Schnittstellen in `java.io` ergänzen die Datenstreams, so daß ein komplettes Ein-/Ausgabesystem bereitgestellt wird. Drei davon werden im folgenden beschrieben.

Die Klasse `File` abstrahiert eine Datei auf plattformunabhängige Weise. Mit einem Dateinamen kann sie auf Anfragen über

Typ, Status und Eigenschaften einer Datei oder eines Verzeichnisses im Dateisystem reagieren.

Anhand einer Datei, eines Dateinamens oder eines Zugriffsmodus (»r« oder »rw«) wird eine `RandomAccessFile` erzeugt. Sie umfaßt Implementierungen von `DataInput` und `DataOutput` in einer Klasse, jeweils auf »Zufallszugriff« auf eine Datei im Dateisystem abgestimmt. Zusätzlich zu diesen Schnittstellen bietet `RandomAccessFile` bestimmte herkömmliche Einrichtungen nach Unix-Art, z.B. `seek()` zum Suchen eines beliebigen Punkts in einer Datei.

Die Klasse `StreamTokenizer` greift einen Eingabestream (oder Reader) heraus und erzeugt daraus eine Folge von Token. Durch Überladen verschiedener darin enthaltener Methoden in Ihren Subklassen können Sie starke lexikale Parser erstellen.

In der API-Beschreibung Ihres Java-Releases finden Sie (online) weitere Informationen über diese Klassen.

Zusammenfassung

Heute haben Sie das allgemeine Konzept von Datenstreams gelernt und Beispiele mit Eingabestreams und Readern auf der Grundlage von Byte-Arrays, Dateien, Pipes, anderen Datenstreamfolgen und Stringpuffern sowie Eingabefiltern, Dateneingaben, Zeilennummerierung und Zurückschieben von Zeichen durchgearbeitet.

Sie haben auch die Gegenstücke dazu - die Ausgabestreams und Writer für Bytearrays, Dateien, Pipes und Ausgabefilter zum Schreiben typisierter Daten und Ausgabefilter - kennengelernt.

Sie haben sich in dieser Lektion Kenntnisse über die grundlegenden Methoden aller Datenstreams (z.B. `read()` und `write()`) und einige spezielle Methoden angeeignet. Sie haben das Auffangen (`catch()`) von Ausnahmen, insbesondere `EOFException`, gelernt.

Sie haben gelernt, mit den doppelt nützlichen Schnittstellen `DataInput` und `DataOutput` umzugehen, die den Kern von `RandomAccessFile` bilden.

Java-Datenstreams bieten eine starke Grundlage, auf der Sie Multithreading-/Streaming-Schnittstellen der komplexesten Art entwickeln können, die in Browsern (z.B. Microsoft Internet Explorer oder Netscape Navigator) interpretiert werden. Die höheren Internet-Protokolle und -Dienste, für die Sie künftig Ihre Applets schreiben können, sind im Prinzip nur durch Ihre Vorstellungskraft beschränkt.

Fragen und Antworten

Frage:

In einem früheren `read()`-Beispiel haben Sie meiner Meinung nach mit der Variablen `byteOrMinus1` etwas Plumpes angestellt. Gibt es dafür keine bessere Art? Und falls nicht, warum haben Sie in einem späteren Abschnitt die Umwandlung empfohlen?

Antwort:

Stimmt, diese Anweisungen haben wirklich etwas Schwerfälliges an sich. Man ist versucht, statt dessen etwa folgenden Code zu schreiben:

```
while ((b = (byte) s.read()) != -1) {
    . . . // Verarbeite Byte b
}
```

Das Problem bei dieser Kurzform entsteht, wenn `read()` den Wert `0xFF` (0377) zurückgibt. Da dieser Wert ein Vorzeichen erhält, bevor der Test ausgeführt wird, erscheint er genauso wie der ganzzahlige Wert -1, der das Datenstreamende bezeichnet. Nur durch Speichern dieses Werts in einer getrennten ganzzahligen Variablen und späteres Umwandeln erreicht man das gewünschte Ergebnis. Ich habe die Umwandlung in `byte` aus Konsistenzgründen empfohlen. Das Speichern ganzzahliger Werte in Variablen mit korrekter Größe entspricht immer einem guten Stil (abgesehen davon sollte `read()` hier eine `byte`- Größe zurückgeben und für das Datenstreamende eine Ausnahme auswerfen).

Frage:

Wozu soll `available()` nützlich sein, wenn es manchmal die falsche Antwort ausgibt?

Antwort:

Erstens muß man zugeben, daß es bei vielen Datenstreams richtig reagiert. Zweitens kann seine Implementierung bei manchen Netzdatenstreams eine spezielle Anfrage senden, um bestimmte Informationen aufzudecken, die Sie andernfalls

nicht einholen können (z.B. die Größe einer über ftp übertragenen Datei). Würden Sie einen Verlaufs balken für das Downloading oder die Übertragung von Dateien anzeigen, gäbe `available()` beispielsweise die Gesamtgröße der Übertragung zurück bzw. andernfalls 0, was für Sie und Ihre Benutzer sichtbar wäre.

Frage:

Können Sie mir ein gutes Beispiel für die Verwendung des Schnittstellenpaars `DataInput/DataOutput` geben?

Antwort:

Eine übliche Verwendung dieses Schnittstellenpaars ist, wenn sich Objekte selbst zum Speichern oder Befördern über das Netz vorbereiten. Jedes Objekt implementiert Lese- und Schreibmethoden anhand dieser Schnittstellen, so daß sie sich selbst effektiv in einen Datenstream umwandeln, der später am anderen Ende als Kopie des Originalobjekts wiederhergestellt werden kann. Dieser Prozeß kann ab Version 1.1 über die neuen Ein- und Ausgabedatenstreams für Objekte automatisiert werden.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Tag 18

Kommunikation über das Internet

Eines der bemerkenswerteren Dinge an Java ist seit seiner Einführung, in welchem Maß die Sprache für das Internet geeignet ist. Wie Sie sich bestimmt vom ersten Tag her erinnern werden, wurde Java ursprünglich als Sprache zur Steuerung eines Netzes interaktiver Geräte mit dem Namen Star7 entwickelt. Duke - das animierte Maskottchen von JavaSoft - war der Star dieser Geräte.

Java's Klassenbibliothek beinhaltet das Paket `java.net`, das es Ihren Java-Programmen ermöglicht, über ein Netzwerk zu kommunizieren. Das Paket bietet eine plattformübergreifende Abstraktionsebene für einfache Netzwerkoperationen, darunter Verbindung zu Dateien aufzubauen und diese zu übertragen. Dazu werden Standard- Web-Protokolle verwendet und elementare Sockets, wie sie von Unix her bekannt sind, erzeugt.

In Verbindung mit den Eingabe- und Ausgabestreams, die Sie gestern kennengelernt haben, wird das Lesen und Schreiben von Dateien über ein Netzwerk genauso einfach, wie es von einer lokalen Festplatte ist.

Heute werden Sie einige Applikationen schreiben, die in der Lage sind, über ein Netzwerk zu kommunizieren. Außerdem werden Sie lernen, warum es bedeutend schwieriger ist, dasselbe mit einem Applet zu tun. Sie werden ein Programm erstellen, das ein Dokument über das World Wide Web laden kann, und werden untersuchen, wie Client/-Server-Programme erzeugt werden.

Netzwerkprogrammierung in Java

Dieser Abschnitt beschreibt zwei einfache Wege, wie Sie mit Systemen im Netz kommunizieren können:

- `getInputStream()`, eine Methode, die eine Verbindung zu einer URL herstellt und das Einholen von Daten über diese Verbindung ermöglicht
- Die Socket-Klassen `Socket` und `ServerSocket`, die das Öffnen von Standard-Socket-Verbindungen zu Hosts und das Lesen und Schreiben über solche Verbindungen ermöglichen

Öffnen von Web-Verbindungen

Anstatt den Browser lediglich aufzufordern, den Inhalt einer Datei zu laden, möchten Sie vielleicht den Inhalt der Datei in Ihrem Applet benutzen. Ist die betreffende Datei im Web gespeichert und über die üblichen URL-Formen (`http`, `ftp` usw.) zugänglich, können Sie die `URL`-Klasse benutzen, um die Datei in Ihrem Applet zu verwenden.

Beachten Sie, daß Applets aus Sicherheitsgründen nur zurück zu dem gleichen Host, von dem sie ursprünglich geladen wurden, Verbindungen herstellen können. Das bedeutet beispielsweise bei einem Applet, das auf einem System namens `www.myhost.com` gespeichert ist, daß Ihr Applet nur mit diesem Host (und dem gleichen Hostnamen, deshalb vorsichtig mit Aliasnamen!) eine Verbindung herstellen kann. Befindet sich eine Datei, die das Applet abrufen möchte, auf dem gleichen System, sind `URL`-Verbindungen die einfachste Möglichkeit, dies zu erreichen.

Diese Sicherheitseinschränkung wird die Art und Weise, in der Sie Applets bis jetzt geschrieben und getestet haben, ändern. Da wir uns noch nicht mit Netzverbindungen beschäftigt haben, war es uns möglich, alle Tests auf der lokalen Platte durch einfaches Öffnen der `HTML`-Dateien oder mit dem Werkzeug zum Betrachten des Applets durchzuführen. Dies ist mit Applets, die Netzverbindungen öffnen, nicht möglich. Damit diese Applets richtig funktionieren, müssen Sie eines von zwei Dingen tun:

- Ihren Browser auf der gleichen Maschine laufen lassen, auf der Ihr Web-Server läuft. Wenn Sie keinen Zugriff auf Ihren Web-Server haben, besteht häufig die Möglichkeit, einen Web-Server auf Ihrer lokalen Maschine zu installieren und damit zu arbeiten.
- Um sie zu testen, laden Sie jedes Mal Ihre Klasse und `HTML`-Dateien auf Ihren Web-Server. Verwenden Sie dann den eigentlichen `URL` der `HTML`-Datei, anstatt »Open File« (Datei öffnen) zum Testen Ihrer Applets.

Sie werden schon merken, ob Ihr Applet und die Verbindung, die es öffnet, auf dem gleichen Server sind. Bei dem Versuch, ein Applet oder eine Datei von unterschiedlichen Servern zu laden, erhalten Sie, zusammen mit anderen auf

Ihrem Bildschirm oder der Java-Konsole ausgegebenen Fehlermeldungen, eine Sicherheitsausnahme.

Beschäftigen wir uns jetzt mit den Methoden und Klassen zum Laden von Dateien aus dem Web.

openStream()

Die URL-Klasse definiert eine Methode namens `openStream()`, die eine Netzverbindung mit einem bestimmten URL öffnet (eine HTTP-Verbindung für Web-URLs, eine FTP-Verbindung für FTP-URLs usw.) und eine Instanz der Klasse `InputStream` (Teil des `java.io`-Pakets) ausgibt. Wenn Sie diesen Stream in einen `DataInputStream` (mit einem `BufferedInputStream` in der Mitte, um die Leistung zu steigern) konvertieren, können Sie Zeichen und Zeilen aus diesem Stream lesen. Die folgenden Zeilen öffnen beispielsweise eine Verbindung zu der URL, die in der Variablen `theURL` gespeichert ist, und lesen dann den Inhalt jeder Zeile der Datei und geben ihn auf dem Standardausgabegerät aus:

```
try {
    InputStream in = theURL.openStream();
    DataInputStream data = new DataInputStream(new BufferedInputStream(in);

    String line;
    while ((line = data.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException e) {
    System.out.println("IO Error: " + e.getMessage());
}
```



Sie müssen alle Zeilen zwischen eine `try...catch`-Anweisung setzen, um erzeugte `IOExceptions` zu berücksichtigen. `IOExceptions` und die `try...catch`-Anweisung wurden am Tag 16 behandelt.

Das folgende Beispiel eines Applets nutzt die `openStream()`-Methode, um eine Verbindung zu einem Web-Standort herzustellen. Dann wird über diese Verbindung eine Datei (»Der Rabe« von Edgar Allen Poe) gelesen und in einem Textbereich angezeigt. Listing 18.1 enthält den Code; das Ergebnis nach dem Lesen der Datei sehen Sie in Abbildung 18.1.

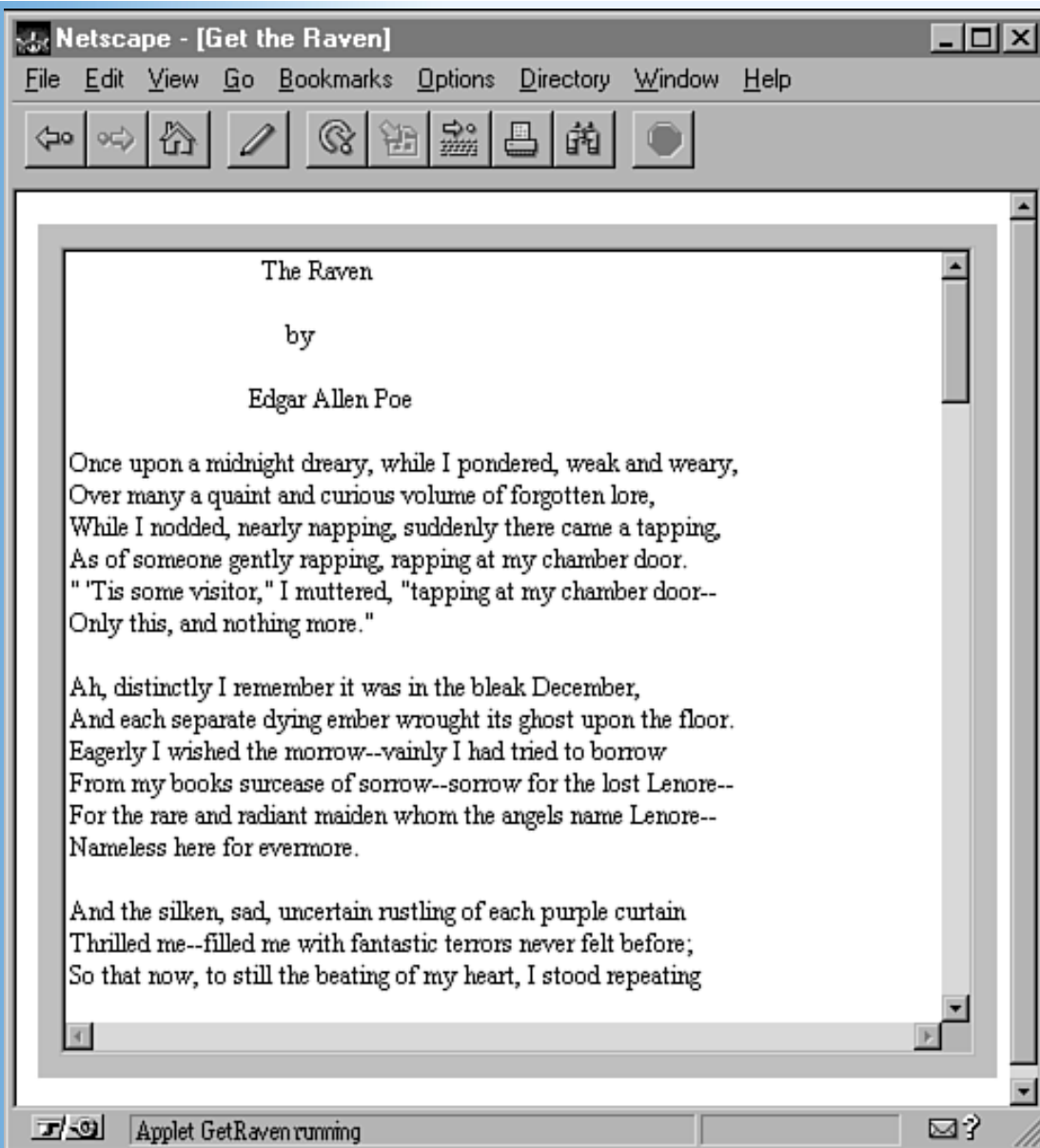


Abbildung 18.1:
Das GetRaven-Applet

Hierzu ein äußerst wichtiger Hinweis: Wenn Sie diesen Code wie geschrieben kompilieren, funktioniert er nicht - und Sie erhalten eine Sicherheitsausnahme. Der Grund dafür ist, daß dieses Applet eine Verbindung zu dem Server www.lne.com zum Holen der Datei `raven.txt` öffnet. Wenn Sie dieses Applet kompilieren und damit arbeiten, läuft dieses Applet nicht auf www.lne.com (es sei denn, Sie sind »Ich« und kennen somit das Problem bereits). Bevor Sie dieses Applet kompilieren können, müssen Sie unbedingt die Zeile 18 so verändern, daß sie auf eine Kopie von `raven.txt` auf Ihrem Web-Server verweist und Ihr Applet und Ihre HTML-Dateien auf dem gleichen Server installieren (Sie können `raven.txt` von der CD oder von der oben angegebenen URL holen).

Alternativ dazu können Sie mit Ihrem Browser zu der URL <http://www.lne.com/Web/JavaProf/GetRaven.html> gehen. Diese Webseite lädt genau dieses Applet und sorgt für korrektes Runterladen der Datei. Da sich sowohl das Applet als auch die Textdatei auf dem gleichen Server befinden, funktioniert alles bestens.

Listing 18.1: Der komplette Quelltext der GetRaven-Klasse

```

1: import java.awt.*;
2: import java.io.DataInputStream;
3: import java.io.BufferedInputStream;
4: import java.io.IOException;
5: import java.net.URL;
6: import java.net.URLConnection;
7: import java.net.MalformedURLException;
8:
9: public class GetRaven extends java.applet.Applet implements Runnable {

```

```

10:  URL theURL;
11:  Thread runner;
12:  TextArea ta = new TextArea("Getting text...");
13:
14:  public void init() {
15:      setLayout(new GridLayout(1,1));
16:
17:      // DIESEN TEXT VOR DER KOMPILIERUNG ÄNDERN!!!
18:      String url = "http://www.lne.com/Web/JavaProf/raven.txt";
19:      try { this.theURL = new URL(url); }
20:      catch ( MalformedURLException e) {
21:          System.out.println("Bad URL: " + theURL);
22:      }
23:      add(ta);
24:  }
25:
26:  public Insets insets() {
27:      return new Insets(10,10,10,10);
28:  }
29:
30:  public void start() {
31:      if (runner == null) {
32:          runner = new Thread(this);
33:          runner.start();
34:      }
35:  }
36:
37:  public void stop() {
38:      if (runner != null) {
39:          runner.stop();
40:          runner = null;
41:      }
42:  }
43:
44:  public void run() {
45:      URLConnection conn = null;
46:      DataInputStream data = null;
47:      String line;
48:      StringBuffer buf = new StringBuffer();
49:
50:      try {
51:          conn = this.theURL.openConnection();
52:          conn.connect();
53:          ta.setText("Connection opened...");
54:          data = new DataInputStream(new BufferedInputStream(
55:              conn.getInputStream()));
56:          ta.setText("Reading data...");
57:          while ((line = data.readLine()) != null) {
58:              buf.append(line + "\n");
59:          }
60:          ta.setText(buf.toString());
61:      }
62:      catch (IOException e) {
63:          System.out.println("IO Error:" + e.getMessage());
64:      }
65:  }
66:  }

```

Die init()-Methode (Zeilen 14 bis 24) setzt die URL und richtet den Textbereich ein, in dem diese Datei angezeigt wird.

Die URL könnte leicht über einen HTML-Parameter an das Applet abgegeben werden; hier wurde er der Einfachheit halber hart kodiert. Da es einige Zeit dauern kann, bis die Datei über das Netz geladen wird, stellen Sie diese Routine in einen eigenen Thread und benutzen die Ihnen inzwischen bestens bekannten Methoden `start()`, `stop()` und `run()`, um diesen Thread zu steuern. Innerhalb von `run()` (Zeilen 44 bis 64) findet die eigentliche Arbeit statt. Hier initialisieren Sie mehrere Variablen und öffnen dann die Verbindung zu der URL (mit der `openStream()`-Methode in Zeile 50). Ist die Verbindung aufgebaut, richten Sie in den Zeilen 51 bis 55 einen Eingabestream ein, von dem zeilenweise gelesen wird. Das Ergebnis wird in eine Instanz von `StringBuffer` (das ist eine änderbare Zeichenkette) gestellt. Ich stelle die gesamte Arbeit in einen Thread, da der Verbindungsaufbau und das Lesen der Datei - insbesondere über langsamere Verbindungen, einige Zeit in Anspruch nehmen kann. Parallel zum Laden der Datei sind möglicherweise andere Aktivitäten in dem Applet auszuführen.

Nachdem alle Daten gelesen wurden, konvertiert Zeile 60 das `StringBuffer`-Objekt in eine echte Zeichenkette und stellt das Ergebnis in den Textbereich.

Bezüglich dieses Beispiels ist noch etwas anderes zu beachten: nämlich daß der Teil des Codes, der eine Netzverbindung geöffnet, aus der Datei gelesen und eine Zeichenkette erstellt hat, zwischen eine `try...catch`-Anweisung gestellt wird. Tritt während des Versuchs, die Datei zu lesen oder zu verarbeiten, ein Fehler auf, ermöglicht diese Anweisung die Fehlerbehandlung, ohne daß das gesamte Programm abstürzt (in diesem Fall endet das Programm mit einem Fehler, weil ansonsten wenig getan werden kann, wenn das Applet die Datei nicht lesen kann). Mit `try...catch` können Sie Ihrem Applet die Möglichkeit geben, auf Fehler zu reagieren und diese entsprechend zu behandeln.

Sockets

Für vernetzte Anwendungen, die über das hinausgehen, was die Klassen `URL` und `URLConnection` bieten (z.B. für andere Protokolle oder allgemeinere vernetzte Anwendungen), bietet Java die Klassen `Socket` und `ServerSocket` als Abstraktion von standardmäßigen TCP-Socket-Programmiertechniken.



Java bietet ebenfalls Möglichkeiten der Verwendung von Datagram-Sockets (UDP, User Datagram Protocol), auf die ich hier allerdings nicht eingehen werde. Wenn Sie daran interessiert sind, mit Datagrammen zu arbeiten, finden Sie entsprechende Informationen in der API-Dokumentation des `java.net`-Pakets.

Die `Socket`-Klasse bietet eine clientseitige Socket-Schnittstelle, die mit Unix-Standard-Sockets vergleichbar ist. Um eine Verbindung herzustellen, legen Sie eine neue Instanz von `Socket` an (wobei der `hostname` der Host ist, zu dem die Verbindung herzustellen, und `portnum` die Portnummer ist):

```
Socket connection = new Socket(hostname, portnum);
```



Auch wenn Sie Sockets in einem Applet verwenden, unterliegen Ihre Applets nach wie vor den Sicherheitseinschränkungen, die Sie daran hindern, eine Verbindung zu einem anderen als dem System, von dem das Applet geladen wird, herzustellen.

Nachdem Sie den `Socket` geöffnet haben, können Sie Ein- und Ausgabestreams verwenden, um über diesen `Socket` zu lesen und zu schreiben:

```
DataInputStream in = new DataInputStream(
new BufferedInputStream(connection.getInputStream()));
DataOutputStream out= new DataOutputStream(
new BufferedOutputStream(connection.getOutputStream()));
```

Zum Schluß müssen Sie den `Socket` schließen (dadurch werden auch alle Ein- und Ausgabestreams geschlossen, die Sie für diesen `Socket` eingerichtet haben):

```
connection.close();
```

Server-seitige Sockets funktionieren auf ähnliche Weise, mit Ausnahme der `accept()`-Methode. Ein `ServerSocket` richtet sich nach einem TCP-Port, um eine Client-Verbindung aufzubauen; wenn sich ein Client mit diesem Port verbindet,

akzeptiert die `accept()`-Methode eine Verbindung von diesem Client. Durch Verwendung von Client- und Server-Sockets können Sie Anwendungen entwickeln, die miteinander über das Netz kommunizieren.

Um einen Server-Socket zu erstellen und an einen Port anzubinden, legen Sie eine neue Instanz von `ServerSocket` mit der Portnummer an:

```
ServerSocket sconnection = new ServerSocket(8888);
```

Um diesen Port zu bedienen (und bei Anfrage eine Verbindung von Clients entgegenzunehmen), benutzen Sie die `accept()`-Methode:

```
sconnection.accept();
```

Sobald die Socket-Verbindung aufgebaut ist, können Sie die Ein- und Ausgabestreams verwenden, um vom Client zu lesen und zu schreiben.

Im nächsten Abschnitt, »Trivia: Ein einfacher Socket-Client und -Server«, gehen wir einige Codes durch, um eine einfache Socket-basierte Anwendung zu realisieren.

In der Version 1.02 von Java bieten die Klassen `Socket` und `ServerSocket` eine einfache abstrakte Socket-Implementierung. Sie können neue Instanzen dieser Klassen zum Aufbau oder Akzeptieren von Verbindungen anlegen und zur Weiter- oder Rückgabe von Daten von einem Client an einen Server.

Das Problem entsteht bei dem Versuch, das Socket-Verhalten von Java zu erweitern oder zu ändern. Die Klassen `Socket` und `ServerSocket` im `java.net`-Paket sind `Final`- Klassen, was bedeutet, daß Sie von diesen Klassen keine Subklassen erzeugen können. Um das Verhalten der Socket-Klassen zu erweitern - beispielsweise um es Netzverbindungen zu ermöglichen, über eine Firewall oder einen Proxy zu arbeiten -, können Sie die abstrakten Klassen `SocketImpl` und die Schnittstelle `SocketImplFactory` verwenden, um eine neue Transportebene der Socket-Implementierung zu erstellen. Dieses Design stimmt mit dem ursprünglichen Konzept für die Java-Socket-Klassen überein: es diesen Klassen zu ermöglichen, mit unterschiedlichen Transportmechanismen auf andere Systeme portierbar zu sein. Das Problem dieses Mechanismus besteht darin, daß, während er in einfachen Fällen funktioniert, er es Ihnen aber nicht ermöglicht, zusätzlich zu TCP noch andere Protokolle hinzuzufügen (z.B. einen Verschlüsselungsmechanismus wie SSL zu realisieren) oder mehrere Socket-Implementierungen zur Java-Laufzeit zu haben.

Deshalb wurden Sockets nach Java 1.02 so geändert, daß die Klassen `Socket` und `ServerSocket` »nicht final« und erweiterbar sind. Sie können jetzt Subklassen dieser Klassen mit Java 1.1 erstellen, die entweder die Standard-Socket-Implementierung benutzen oder eine von Ihnen selbst kreierte. Dies gestaltet die Netzwerkfähigkeiten wesentlich flexibler.

Darüber hinaus wurden dem `java.net`-Paket verschiedene neue Features hinzugefügt:

- Neue Socket-Optionen, auf den Socket-Optionen von BSD basierend (beispielsweise `TCP_NODELAY`, `IP_MULTICAST_LOOP`, `SO_BINDADDR`)
- Viele neue Subklassen der `SocketException`-Klasse, um Netzfehler genauer als in Java 1.02 darzustellen (z.B. `NoRouteToHostException` oder `Connect-Exception`)

Trivia: Ein einfacher Socket-Client und -Server

Den Abschluß des Themas Netzwerkprogrammierung mit Java bildet das Beispiel eines Java-Programms, das die Socket-Klasse zur Realisierung einer einfachen netzbasierten Anwendung namens Trivia benutzt.

Trivia arbeitet folgendermaßen: Das Server-Programm wartet geduldig auf die Herstellung einer Verbindung eines Clients. Wird die Verbindung von einem Client hergestellt, übermittelt der Server eine Frage und wartet auf die Reaktion. Am anderen Ende erhält der Client die Frage und veranlaßt den Benutzer zur Antwort. Der Benutzer gibt eine Antwort ein, die an den Server zurückübermittelt wird. Der Server überprüft dann, ob die Antwort richtig ist, und informiert den Benutzer. Der Server faßt noch einmal nach, indem er den Client fragt, ob er eine andere Frage möchte. Falls ja, wird der Prozeß wiederholt.

Trivia entwerfen

Im allgemeinen erweist es sich als zweckdienlich, bevor Sie damit beginnen, in umfangreichem Maße Code zu produzieren, einen kurzen vorläufigen Entwurf anzufertigen. Schauen wir uns also zuerst einmal an, was wir für den Trivia-Server und -Client benötigen. Server-seitig brauchen Sie ein Programm, das einen spezifischen Port der

Hostmaschine hinsichtlich Client-Verbindungen überwacht. Wird ein Client entdeckt, wählt der Server eine Zufallsfrage und übermittelt sie über diesen spezifischen Port an den Client. Der Server gibt dann einen Wartestatus ein, bis er erneut eine Reaktion vom Client verzeichnet. Erhält der Server eine Antwort vom Client, überprüft er sie und gibt dem Client bekannt, ob die Antwort richtig oder falsch ist. Anschließend fragt der Server den Client, ob er eine weitere Frage wünscht, woraufhin er bis zur Antwort des Clients einen weiteren Wartestatus eingibt. Abschließend wiederholt der Server entweder den Prozeß, indem er eine weitere Frage stellt, oder beendet die Verbindung mit dem Client. Zusammenfassend führt der Server die folgenden Aufgaben aus:

1. Warten auf die Verbindungsherstellung eines Clients
2. Akzeptieren der Client-Verbindung
3. Übermittlung einer Zufallsfrage an den Client
4. Warten auf eine Antwort vom Client
5. Überprüfung der Antwort und Information des Clients
6. Anfrage an den Client, ob er eine weitere Frage wünscht
7. Warten auf eine Antwort vom Client
8. Falls erforderlich, erneutes Ansetzen bei Schritt 3.

Client-seitig ist dieses Trivia-Beispiel eine Anwendung, die von einer Befehlszeile aus arbeitet (auf diese Art leichter zu demonstrieren). Der Client ist für die Verbindungsherstellung zum Server zuständig und wartet auf eine Frage. Bei Erhalt einer Frage vom Server zeigt der Client diese dem Benutzer an und gibt dem Benutzer die Möglichkeit zur Eingabe einer Antwort. Diese Antwort wird an den Server zurückübermittelt, und der Client wartet wieder auf die Reaktion des Servers. Der Client zeigt dem Benutzer die Antwort des Servers an und ermöglicht dem Benutzer zu bestätigen, ob er eine weitere Frage wünscht. Der Client sendet dann die Antwort des Benutzers an den Server und beendet die Verbindung, falls der Benutzer keine weiteren Fragen wünscht. Die hauptsächlichen Aufgaben des Clients sind:

1. Herstellen der Verbindung zum Server
2. Warten auf eine zu übermittelnde Frage
3. Anzeige der Frage und Eingabe der Antwort des Benutzers
4. Übermittlung der Antwort an den Server
5. Warten auf Antwort vom Server
6. Anzeige der Antwort des Servers und Veranlassung des Benutzers zur Bestätigung einer weiteren Frage
7. Übermittlung der Antwort des Benutzers an den Server
8. Falls erforderlich, erneutes Ansetzen bei Schritt 2.

Trivia-Server implementieren

Der Server bildet den wesentlichsten Bestandteil bei den Trivia-Beispielen. Das Trivia- Server-Programm heißt TriviaServer. Hier die in der TriviaServer-Klasse definierten Instanzvariablen:

```
private static final int PORTNUM = 1234;
private static final int WAITFORCLIENT = 0;
private static final int WAITFORANSWER = 1;
private static final int WAITFORCONFIRM = 2;
private String[] questions;
private String[] answers;
private ServerSocket serverSocket;
private int numQuestions;
private int num = 0;
private int state = WAITFORCLIENT;
private Random rand = new Random(System.currentTimeMillis());
```

Die Konstanten WAITFORCLIENT, WAITFORANSWER und WAITFORCONFIRM sind allesamt Statuskonstanten,

die der Definition unterschiedlicher Status, in denen sich der Server befinden kann, dienen. Den Einsatz dieser Konstanten sehen Sie gleich. Die Frage- und Antwortvariablen sind Zeichenketten-Arrays zur Speicherung der Fragen und Antworten. Die `serverSocket`-Instanzvariable richtet sich nach der Server-Socket-Verbindung. `numQuestions` wird zur Speicherung der Gesamtanzahl der Fragen benutzt, wobei `num` die Anzahl der aktuell gestellten Fragen wiedergibt. Die `state`-Variable verfügt über den aktuellen Status des Servers wie von den drei Statuskonstanten (`WAITFORCLIENT`, `WAITFORANSWER` und `WAITFORCONFIRM`) festgelegt. Und die `rand`-Variable wird dazu verwendet, zufällig Fragen auszuwählen.

Der `TriviaServer`-Konstruktor macht nicht viel, mit Ausnahme der Erstellung eines `ServerSocket` anstatt eines `DatagramSocket`. Schauen Sie sich's an:

```
public TriviaServer() {
    super("TriviaServer");
    try {
        serverSocket = new ServerSocket(PORTNUM);
        System.out.println("TriviaServer up and running...");
    }
    catch (IOException e) {
        System.err.println("Exception: couldn't create socket");
        System.exit(1);
    }
}
```

Der größte Teil der Aktionen spielt sich in der `run()`-Methode in der `TriviaServer`- Klasse ab. In Anschluß sehen Sie den Quellcode für die `run()`-Methode.

```
public void run() {
    Socket clientSocket;

    // Initialisieren der Fragen-/Antwort-Arrays
    if (!initQnA()) {
        System.err.println("Error: couldn't initialize questions and answers");
        return;
    }

    // Nach Clients suchen und Trivia-Fragen stellen
    while (true) {
        // Auf Client warten
        if (serverSocket == null)
            return;
        try {
            clientSocket = serverSocket.accept();
        }
        catch (IOException e) {
            System.err.println("Exception: couldn't connect to client socket");
            System.exit(1);
        }

        // Fragen-/Antwortverarbeitung durchführen
        try {
            DataInputStream is = new DataInputStream(new
                BufferedInputStream(clientSocket.getInputStream()));
            PrintStream os = new PrintStream(new
                BufferedOutputStream(clientSocket.getOutputStream()), false);
            String inLine, outLine;

            // Serveranfrage ausgeben
            outLine = processInput(null);
            os.println(outLine);
            os.flush();
        }
        catch (IOException e) {
            System.err.println("Exception: couldn't process input");
            System.exit(1);
        }
    }
}
```

```

        // Verarbeitung und Ausgabe der Benutzereingabe
        while ((inLine = is.readLine()) != null) {
            outLine = processInput(inLine);
            os.println(outLine);
            os.flush();
            if (outLine.equals("Bye."))
                break;
        }

        // Aufräumen
        os.close();
        is.close();
        clientSocket.close();
    }
    catch (Exception e) {
        System.err.println("Exception: " + e);
        e.printStackTrace();
    }
}
}
}

```

Durch Aufrufen von `initQnA()` initialisiert die `run()`-Methode zunächst die Fragen und Antworten. Über die `initQnA()`-Methode werden Sie gleich mehr erfahren. Danach wird eine Endlos-while-Schleife gestartet, die auf eine Client-Verbindung wartet. Stellt ein Client die Verbindung her, werden die entsprechenden I/O-Streams erzeugt und die Kommunikation durch die `processInput()`-Methode verarbeitet. `processInput()` ist unser nächstes Thema. `processInput()` verarbeitet kontinuierlich Client- Antworten und sorgt für das Stellen neuer Fragen, bis der Client sich entschließt, keine weiteren Fragen mehr zu erhalten. Dies wird vom Server entsprechend durch Übermitteln der Zeichenkette »Bye.« bestätigt. Die `run()`-Methode sorgt anschließend dafür, daß die Streams und der Client-Socket geschlossen werden.

Die `processInput()`-Methode richtet sich nach dem Server-Status und stellt die Logik des gesamten Fragen-/Antwortprozesses. Im Anschluß finden Sie den Quellcode für `processInput`.

```

String processInput(String inStr) {
    String outStr = null;

    switch (state) {
        case WAITFORCLIENT:
            // Eine Frage stellen
            outStr = questions[num];
            state = WAITFORANSWER;
            break;

        case WAITFORANSWER:
            // Die Antwort prüfen
            if (inStr.equalsIgnoreCase(answers[num]))
                outStr = "That's correct! Want another? (y/n)";
            else
                outStr = "Wrong, the correct answer is " + answers[num] +
                    ". Want another? (y/n)";
            state = WAITFORCONFIRM;
            break;

        case WAITFORCONFIRM:
            // Prüfen, ob eine weitere Frage gewünscht wird
            if (inStr.equalsIgnoreCase("Y")) {
                num = Math.abs(rand.nextInt()) % questions.length;
                outStr = questions[num];
                state = WAITFORANSWER;
            }
            else {

```



```

        outStr = "Bye.";
        state = WAITFORCLIENT;
    }
    break;
}
return outStr;
}

```

Als erstes ist bei der processInput()-Methode die lokale Variable outStr zu beachten. Der Wert dieser Zeichenkette wird in der Run-Methode an den Client zurückgesandt, wenn processInput() antwortet. Beachten Sie also, wie processInput() die lokale Variable outStr benutzt, um Informationen an den Client zurückzuführen.

In FortuneServer stellt der Status WAITFORCLIENT den Server im Status leer und auf eine Client-Verbindung wartend dar. Das bedeutet also, daß jedes case-Statement in der processInput()-Methode den Server in dem Status, den er gerade verläßt, darstellt. Das case-Statement WAITFORCLIENT wird beispielsweise eingegeben, wenn der Server den Status WAITFORCLIENT gerade verlassen hat. Anders ausgedrückt hat ein Client gerade eine Verbindung zum Server hergestellt. In diesem Fall setzt der Server die Ausgabezeichenkette auf die aktuelle Frage und den Status auf WAITFORANSWER.

Verläßt der Server den Status WAITFORANSWER, bedeutet dies, daß der Client mit einer Antwort reagiert hat. processInput() vergleicht die Antwort des Clients mit der richtigen Antwort und setzt dementsprechend die Ausgabezeichenkette. Anschließend setzt sie den Status auf WAITFORCONFIRM.

Im WAITFORCONFIRM-Status wartet der Server auf eine Bestätigungsantwort vom Client. In der Methode processInput() zeigt das case-Statement WAITFORCONFIRM an, daß der Server den Status verläßt, da der Client mit einer Bestätigung (ja oder nein) geantwortet hat. Hat der Client die Frage mit y bejaht, wählt processInput() eine neue Frage und setzt den Status wieder auf WAITFORANSWER. Andernfalls gibt der Server Bye. an den Client aus und setzt den Status erneut auf WAITFORCLIENT, um auf eine neue Client-Verbindung zu warten.

Die Trivia-Fragen und -Antworten sind in einer Textdatei namens QnA.txt gespeichert und dort zeilenweise mit Fragen und Antworten im Wechsel aufgebaut. Wechselweise bedeutet, daß auf jede Frage eine entsprechende Antwort auf der nächsten Zeile folgt, woran sich wiederum die nächste Frage anschließt. Hier eine teilweise Auflistung der Datei QnA.txt:

```

What caused the craters on the moon?
meteorites
How far away is the moon (in miles)?
239000
How far away is the sun (in millions of miles)?
93
Is the Earth a perfect sphere?
no
What is the internal temperature of the Earth (in degrees)?
9000

```

Die initQnA()-Methode kümmert sich um das Lesen der Fragen und Antworten aus der Textdatei und deren Speicherung in getrennten String-Arrays. Im folgenden sehen Sie den Quellcode für die initQnA()-Methode.

```

private boolean initQnA() {
    try {
        File inFile = new File("QnA.txt");
        FileInputStream inStream = new FileInputStream(inFile);
        byte[] data = new byte[(int)inFile.length()];

        // Die Fragen und Antworten in ein Array vom Typ byte einlesen
        if (inStream.read(data) <= 0) {
            System.err.println("Error: couldn't read questions and answers");
            return false;
        }
        // Die Anzahl der Fragen/Antworten-Paare ermitteln
        for (int i = 0; i < data.length; i++)
            if (data[i] == (byte)'\n')
                numQuestions++;
    }
}

```



```

        numQuestions /= 2;
        questions = new String[numQuestions];
        answers = new String[numQuestions];

        // Die Fragen und Antworten in String-Arrays einlesen
        int start = 0, index = 0;
        boolean isQ = true;
        for (int i = 0; i < data.length; i++)
            if (data[i] == (byte)'\n') {
                if (isQ) {
                    questions[index] = new String(data, start, i - start - 1);
                    isQ = false;
                }
                else {
                    answers[index] = new String(data, start, i - start - 1);
                    isQ = true;
                    index++;
                }
                start = i + 1;
            }
    }
    catch (FileNotFoundException e) {
        System.err.println("Exception: couldn't find the question file");
        return false;
    }
    catch (IOException e) {
        System.err.println("Exception: I/O error trying to read questions");
        return false;
    }

    return true;
}

```

Die `initQnA()`-Methode setzt zwei Arrays ein und füllt sie abwechselnd mit Zeichenketten aus der `QnA.txt`-Datei: erst eine Frage, dann eine Antwort, jeweils im Wechsel, bis das Dateiende erreicht ist.

Die einzige in `TriviaServer` verbleibende Methode ist `main()`, die lediglich das `Server`-Objekt erzeugt und es mit einem Aufruf an die `start()`-Methode startet:

```

public static void main(String[] args) {
    TriviaServer server = new TriviaServer();
    server.start();
}

```

In Listing 18.2 finden Sie den vollständigen Quelltext der `Server`-Applikation.

Listing 18.2: Der gesamte Quelltext von `TriviaServer.java`

```

1: import java.io.*;
2: import java.net.*;
3: import java.util.Random;
4:
5: public class TriviaServer extends Thread {
6:     private static final int PORTNUM = 1234;
7:     private static final int WAITFORCLIENT = 0;
8:     private static final int WAITFORANSWER = 1;
9:     private static final int WAITFORCONFIRM = 2;
10:    private String[] questions;
11:    private String[] answers;
12:    private ServerSocket serverSocket;
13:    private int numQuestions;

```

```
14:     private int num = 0;
15:     private int state = WAITFORCLIENT;
16:     private Random rand = new Random();
17:
18:     public TriviaServer() {
19:         super("TriviaServer");
20:         try {
21:             serverSocket = new ServerSocket(PORTNUM);
22:             System.out.println("TriviaServer up and running ...");
23:         }
24:         catch (IOException e) {
25:             System.err.println("Exception: couldn't create socket");
26:             System.exit(1);
27:         }
28:     }
29:
30:     public static void main(String[] arguments) {
31:         TriviaServer server = new TriviaServer();
32:         server.start();
33:     }
34:
35:     public void run() {
36:         Socket clientSocket = null;
37:
38:         // Initialisieren der Fragen-/Antwort-Arrays
39:         if (!initQnA()) {
40:             System.err.println("Error: couldn't initialize questions
& and answers");
41:             return;
42:         }
43:
44:         // Nach Clients suchen und Trivia-Fragen stellen
45:         while (true) {
46:             // Auf Client warten
47:             if (serverSocket == null)
48:                 return;
49:             try {
50:                 clientSocket = serverSocket.accept();
51:             }
52:             catch (IOException e) {
53:                 System.err.println("Exception: couldn't connect to
& client socket");
54:                 System.exit(1);
55:             }
56:
57:             // Fragen-/Antwortverarbeitung durchführen
58:             try {
59:                 InputStreamReader isr = new
& InputStreamReader(clientSocket.getInputStream());
60:                 BufferedReader is = new BufferedReader(isr);
61:                 PrintWriter os = new PrintWriter(new
& BufferedOutputStream(clientSocket.getOutputStream()),
& false);
62:                 String outLine;
63:
64:                 // Serveranfrage ausgeben
65:                 outLine = processInput(null);
66:                 os.println(outLine);
67:
```

```

68:                os.flush();
69:
70:                // Verarbeitung und Ausgabe der Benutzereingabe
71:                while (true) {
72:                    String inLine = is.readLine();
73:                    if (inLine.length() > 0) {
74:                        outLine = processInput(inLine);
75:                        os.println(outLine);
76:                        os.flush();
77:                        if (outLine.equals("Bye."))
78:                            break;
79:                    }
80:                }
81:
82:                // Aufräumen
83:                os.close();
84:                is.close();
85:                clientSocket.close();
86:            }
87:            catch (Exception e) {
88:                System.err.println("Exception: " + e);
89:                e.printStackTrace();
90:            }
91:        }
92:    }
93:
94:    private boolean initQnA() {
95:        try {
96:            File inFile = new File("QnA.txt");
97:            FileInputStream inStream = new FileInputStream(inFile);
98:            byte[] data = new byte[(int)inFile.length()];
99:
100:            // Die Fragen und Antworten in ein Array vom Typ byte einlesen
101:            if (inStream.read(data) <= 0) {
102:                System.err.println("Error: couldn't read questions and
103:                answers");
104:                return false;
105:            }
106:
107:            // Die Anzahl der Fragen/Antworten-Paare ermitteln
108:            for (int i = 0; i < data.length; i++)
109:                if (data[i] == (byte)'\n')
110:                    numQuestions++;
111:            numQuestions /= 2;
112:            questions = new String[numQuestions];
113:            answers = new String[numQuestions];
114:
115:            // Die Fragen und Antworten in String-Arrays einlesen
116:            int start = 0, index = 0;
117:            boolean isQ = true;
118:            for (int i = 0; i < data.length; i++)
119:                if (data[i] == (byte)'\n') {
120:                    if (isQ) {
121:                        questions[index] = new String(data, start, i - start
122:                        - 1);
123:                        isQ = false;
124:                    }
125:                    else {

```

```

124:             answers[index] = new String(data, start, i - start -
125:             1);
126:             isQ = true;
127:             index++;
128:         }
129:         start = i + 1;
130:     }
131:     catch (FileNotFoundException e) {
132:         System.err.println("Exception: couldn't find the question file");
133:         return false;
134:     }
135:     catch (IOException e) {
136:         System.err.println("Exception: I/O error trying to read
137:         & questions");
138:         return false;
139:     }
140:     return true;
141: }
142:
143: String processInput(String inStr) {
144:     String outStr = null;
145:
146:     switch (state) {
147:         case WAITFORCLIENT:
148:             // Eine Frage stellen
149:             outStr = questions[num];
150:             state = WAITFORANSWER;
151:             break;
152:
153:         case WAITFORANSWER:
154:             // Die Antwort prüfen
155:             if (inStr.equalsIgnoreCase(answers[num]))
156:                 outStr = "That's correct! Want another? (y/n)";
157:             else
158:                 outStr = "Wrong, the correct answer is " + answers[num] +
159:                 ". Want another? (y/n)";
160:             state = WAITFORCONFIRM;
161:             break;
162:
163:         case WAITFORCONFIRM:
164:             // Prüfen, ob eine weitere Frage gewünscht wird
165:             if (inStr.equalsIgnoreCase("Y")) {
166:                 num = Math.abs(rand.nextInt()) % questions.length;
167:                 outStr = questions[num];
168:                 state = WAITFORANSWER;
169:             }
170:             else {
171:                 outStr = "Bye.";
172:                 state = WAITFORCLIENT;
173:             }
174:             break;
175:     }
176:     return outStr;
177: }
178: }

```

Den Trivia-Client implementieren

Da Client-seitig im Trivia-Beispiel der Benutzer Antworten eingeben und Rückantworten vom Server erhalten muß, stellt sich die Implementierung als Befehlszeilenanwendung unkomplizierter dar. Vielleicht ist das nicht so nett wie ein grafisches Applet, gestaltet es aber sehr einfach, die Entwicklung der Kommunikationsereignisse zu verfolgen. Die Client-Anwendung heißt Trivia.

Die einzige in der Trivia-Klasse definierte Instanzvariable ist PORTNUM, die zur Definition der sowohl vom Client als auch vom Server benutzten Portnummer dient. Es gibt auch nur eine in der Trivia-Klasse definierte Methode: main(). Der Quellcode der main()-Methode ist in Listing 18.3 aufgeführt.

Listing 18.3: Der gesamte Quelltext von Trivia.java

```

1: import java.io.*;
2: import java.net.*;
3:
4: public class Trivia {
5:     private static final int PORTNUM = 1234;
6:
7:     public static void main(String[] arguments) {
8:         Socket socket = null;
9:         InputStreamReader isr = null;
10:        BufferedReader in = null;
11:        PrintWriter out = null;
12:        String address;
13:
14:        // Befehlszeilenargumente für Host-Adresse prüfen
15:        if (arguments.length != 1) {
16:            System.out.println("Usage: java Trivia <address>");
17:            return;
18:        }
19:        else
20:            address = arguments[0];
21:
22:        // Socket und Streams initialisieren
23:        try {
24:            socket = new Socket(address, PORTNUM);
25:            isr = new InputStreamReader(socket.getInputStream());
26:            in = new BufferedReader(isr);
27:            out = new PrintWriter(socket.getOutputStream(), true);
28:        }
29:        catch (IOException e) {
30:            System.err.println("Exception: couldn't create stream socket "
31:                + e.getMessage());
32:            System.exit(1);
33:        }
34:
35:        // Benutzereingabe und Server-Reaktion verarbeiten
36:        try {
37:            StringBuffer str = new StringBuffer(128);
38:            String inStr;
39:            int c;
40:
41:            while ((inStr = in.readLine()) != null) {
42:                System.out.println("Server: " + inStr);
43:                if (inStr.equals("Bye."))
44:                    break;
45:                while ((c = System.in.read()) != '\n')
46:                    str.append((char)c);
47:                System.out.println("Client: " + str);

```



```

48:         out.println(str.toString());
49:         out.flush();
50:         str.setLength(0);
51:     }
52:     // Aufräumen
53:     out.close();
54:     in.close();
55:     socket.close();
56: }
57: catch (IOException e) {
58:     System.err.println("I/O error: "+ e.toString());
59: }
60: }
61: }

```

Was Ihnen vielleicht als erstes bei der main()-Methode auffällt ist, daß sie ein Befehlszeilenargument sucht. Das erforderliche Befehlszeilenargument des Trivia-Clients ist die Server-Adresse wie z.B. thetribe.com. Da es sich hierbei um eine Java-Anwendung und nicht um ein Applet handelt, reicht es nicht aus, nur die Verbindung zurück zum Server, von dem das Applet stammt, herzustellen - es gibt keinen Standard-Server, Sie können also die Verbindung zu einem beliebigen Server herstellen. In der Client-Anwendung müssen Sie die Server-Adresse entweder hart kodieren oder sie als Befehlszeilenargument anfordern. Ich bin kein großer Freund des Hartkodierens, da jede Änderung erneutes Kompilieren notwendig macht. Also das Befehlszeilenargument!



Die meisten Leser werden wahrscheinlich keinen Zugriff auf einen Web-Server haben, der Server-seitige Java-Programme wie die TriviaServer-Applikation ausführt. Unter manchen Betriebssystemen können Sie Serverprogramme testen, indem Sie den Trivia-Server in einem Fenster und den Trivia-Client in einem anderen Fenster ausführen, und die Domäne »localhost« verwenden. Im Anschluß ein Beispiel hierfür:

```
java Trivia "localhost"
```

Dies veranlaßt Java dazu, auf dem lokalen Host - mit anderen Worten dem System, auf dem die Applikation ausgeführt wird - nach einem Server zu suchen, mit dem eine Verbindung aufgebaut werden kann. Abhängig davon, wie Internet-Verbindungen auf Ihrem System eingerichtet wurden, müssen Sie sich eventuell mit dem Internet verbinden, bevor eine erfolgreiche Socket-Verbindung zwischen dem Trivia-Client und dessen Server aufgebaut werden kann.

Ist das Befehlszeilenargument der Server-Adresse gültig (ungleich Null), erstellt die main()-Methode den erforderlichen Socket und I/O-Streams. Anschließend startet sie eine while-Schleife, wobei sie Informationen vom Server verarbeitet und Benutzeranfragen an den Server zurückübermittelt. Wenn der Server die Übermittlung von Informationen beendet (mit Bye.), wird die while-Schleife mit break verlassen, und die main()-Methode sorgt für das Schließen des Sockets und der Streams. Jetzt wissen Sie alles über den Trivia-Client!

Trivia starten

Der Trivia-Server muß laufen, damit der Client arbeiten kann. Dazu müssen Sie zuerst unter Verwendung des Java-Interpreters den Server starten; dies wird folgendermaßen über einen Befehl auf Befehlszeilen-Ebene erreicht:

```
java TriviaServer
```

Auch der Trivia-Client wird über die Befehlszeile gestartet, allerdings müssen Sie eine Server-Adresse als einziges Argument spezifizieren. Nachfolgend ein Beispiel zum Starten des Trivia-Clients und Anschluß an den Server localhost:

```
java Trivia "localhost"
```

Wenn Trivia-Client läuft und einige Fragen beantwortet hat, sollte die Ausgabe etwa so aussehen:

```

Server: Is the Galaxy rotating?
yes
Client: yes
Server: That's correct! Want another? (y/n)
y

```

```
Client: y
Server: Is the Earth a perfect sphere?
no
Client: no
Server: That's correct! Want another? (y/n)
y
Client: y
Server: What caused the craters on the moon?
asteroids
Client: asteroids
Server: Wrong, the correct answer is meteorites. Want another? (y/n)
n
Client: n
Server: Bye.
```

Zusammenfassung

Die Netzwerkprogrammierung hat viele Anwendungen, die Sie in Ihren Anwendungen nutzen können. Sie haben es vielleicht nicht gemerkt, aber das GetRaven-Projekt war ein sehr rudimentärer Webbrowser. Hier wurde über das Internet der Inhalt einer Text-Datei zu einem Java-Programm übertragen und dort angezeigt. Natürlich fehlt jegliche Funktionalität für das HTML-Parsing. JavaSoft hat allerdings einen kompletten Webbrowser in Java geschrieben - HotJava.

Heute haben Sie gelernt, wie Sie URLs, URL-Verbindungen und Eingabestreams gemeinsam verwenden, um Daten aus dem World Wide Web in Ihre Programme zu bekommen.

Sie haben auch gelernt, wie Client- und Server-Programme in Java geschrieben werden und wie ein Server-Programm sich an einen Internet-Port hängt, um auf ein Client-Programm zu warten, das Kontakt aufnimmt.

Fragen und Antworten

Frage:
Wie kann ich eine HTML-Übertragung in einem Java-Applet simulieren?

Antwort:
Derzeit ist das in Applets schwierig. Die beste (und einfachste) Möglichkeit ist die Verwendung der GET-Notation, um den Browser zu veranlassen, den Formularinhalt für Sie einzureichen.

HTML-Formulare können auf zwei Arten übermittelt werden: durch Verwendung der GET-Anfrage oder mit POST. Wenn Sie GET verwenden, werden die Informationen Ihres Formulars in der URL kodiert. Das kann etwa so aussehen:

`http://www.blah.com/cgi-bin/myscript?foo=1&bar=2&name=Laura`

Da das Formular im URL kodiert ist, können Sie ein Java-Applet schreiben, das ein Formular simuliert, Eingaben vom Benutzer anfordern und dann ein neues URL-Objekt mit den Formulardaten erstellen. Dann geben Sie diese URL mit `getAppletContext()`, `showDocument()`, an den Browser weiter. Der Browser übermittelt die Formularergebnisse selbst. Bei einfachen Formularen genügt das.

Frage:
Wie kann ich POST für die Formularübermittlung realisieren?

Antwort:
Sie müssen simulieren, was ein Browser macht, um Formulare mit POST übersenden zu können. Öffnen Sie einen Socket zum Server, und übersenden Sie die Daten. Das sieht etwa so aus (das genaue Format wird vom HTTP bestimmt):

```
POST /cgi-bin/mailto.cgi HTTP/1.0
Content-type: application/x-www-form-urlencoded
Content-length: 36
```

{hier stehen Ihre codierten Formulardaten }

Wenn Sie alles richtig gemacht haben, erhalten Sie die CGI-Formularausgabe vom Server zurück. Dann liegt es an Ihrem

Applet, diese Ausgabe korrekt zu verarbeiten. Im Fall einer Ausgabe in HTML besteht eigentlich keine Möglichkeit, diese Ausgabe an den Browser, in dem Ihr Applet ausgeführt wird, weiterzugeben. Falls Sie eine URL zurückerhalten, können Sie den Browser auf diese URL umleiten.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).

Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 3

Tag 19

JavaBeans und andere fortgeschrittene Features

Von einer Sprache, die so schnell wie Java wächst, kann man sich schnell von der Menge der Klassen, die von JavaSoft angeboten werden, überrollt fühlen. Jedes neue Release der Sprache führt bemerkenswerte Features wie JavaBeans, Java Database Connectivity und 2D-Grafik, ein.

Glücklicher Weise müssen Sie nicht alle Teile der Standard-Klassenbibliothek meistern, bevor Sie nützliche Programme erstellen können. Sie können sich auf die Pakete und Klassen konzentrieren, die Sie in Ihrem Fachbereich benötigen und Ihre Kenntnisse in neuen Bereichen nach Bedarf erweitern.

Heute erhalten Sie eine Einführung in einige der fortgeschritteneren Features, die mit dem aktuellsten Release von Java angeboten werden, darunter auch die folgenden:

- JavaBeans
- Remote Method Invocation
- Java Database Connectivity

Für den größten Teil des heutigen Tages gilt die Direktive, Sie mit dem jeweiligen Thema vertraut zu machen als ersten Schritt, diese Klassen zu verwenden. Allerdings werden Sie auch einige konkrete Projekte zu den Themen Datentransfer und Applet- Browser-Kommunikation finden.

JavaBeans

Schon seit geraumer Zeit verfolgt die Gemeinde der Software-Entwicklung verstärkt den Gedanken wiederverwendbarer Komponenten. Eine Komponente - im allgemeinen Sinn, nicht im Sinne des AWT - ist ein wiederverwendbares Software-Teil, das leicht zur Erstellung von Anwendungen verwendet werden kann, und zwar mit weitaus größerer Entwicklungseffizienz. Dieser Gedanke der Wiederverwendung sorgfältig zusammengestellter Software wurde zu einem gewissen Maße dem Verfahren des Montagebandes entliehen, das während der industriellen Revolution in den Vereinigten Staaten, lange vor der Ära moderner Computer, so populär wurde. Auf Software angewandt bedeutet das, einmalig zu Beginn kleine, wiederverwendbare Komponenten zu bauen und sie dann in höchstmöglichem Maße wiederzuverwenden und damit den gesamten Entwicklungsprozeß zu rationalisieren.



Eine *Software-Komponente* ist ein Stück Software, das in eine diskrete, leicht wiederverwendbare Struktur gekapselt ist.

Obwohl Komponenten-Software ihre Vorteile bietet, muß sich die vollständig wiederverwendbare Software erst noch richtig etablieren. Dies aus einer Vielzahl von Gründen, nicht zuletzt der Tatsache, daß die Software-Industrie im Vergleich zu den Industrien, die sich während der industriellen Revolution etabliert haben, noch sehr jung ist. Es leuchtet ein, daß es einige Zeit dauert, die schwachen Punkte im gesamten Software-Produktionsprozeß auszumerzen. (Wenn Sie meine Einstellung teilen, nehmen Sie die rapiden, in der Welt der Software stattfindenden Veränderungen bereitwillig an und genießen die Tatsache, daß Sie ein Teil von so etwas ähnlichem wie einer Revolution sind - eine Informationsrevolution. Aber ich schweife ab!)

Das vielleicht größte Problem, dem sich Komponenten-Software stellen mußte, ist der große Bereich

ungleichartiger Mikroprozessoren und heute eingesetzter Betriebssysteme. Es gab eine Reihe angemessener Versuche in der Komponenten-Software, die aber immer auf ein spezifisches Betriebssystem limitiert waren. Die VBX- und OCX-Komponentenarchitekturen von Microsoft verzeichneten große Erfolge in der Welt des PC, konnten aber wenig zum Schließen der Lücke zwischen anderen Betriebssystemtypen beitragen. Wirft man das Arbeitsvolumen in die Waagschale, das dazu benötigt wird, um eine inhärente, plattformabhängige Komponententechnologie auf einer Vielzahl von Betriebssystemen zum Laufen zu bringen, macht die ausschließliche Fokussierung auf den PC-Markt von Microsoft Sinn.



Tatsächlich zielt die auf ihrer OCX-Technologie basierte ActiveX-Technologie von Microsoft darauf ab, eine universelle Komponententechnologie bereitzustellen, die Kompatibilität für einen breiten Bereich von Plattformen bietet. Betrachtet man allerdings die Abhängigkeit des ActiveX von dem 32-Bit-Windows-Code, bleibt abzuwarten, wie Microsoft das Problem der Plattformabhängigkeit lösen wird. Vielleicht wartet man nur darauf, daß alle zu Windows 95/NT wechseln?

Bevor es zu der explosionsartigen Entwicklung im Internet-Bereich kam, stellte die Plattformabhängigkeit nicht so ein großes Problem dar. PC-Entwickler machten sich nicht unbedingt allzu viele Gedanken darüber, daß ihre Produkte nicht auf einem Solaris-System laufen konnten. Okay, einige PC-Entwickler gingen auf Nummer sicher und sorgten für Anschluß Ihrer Anwendungen an die Macintosh-Plattform, allerdings oftmals unter beträchtlichem Entwicklungsaufwand. Das gesamte Szenario änderte sich durch den vom Internet erzeugten Schmelztiegel der Betriebssysteme. Daraus resultierte ein erneutes Interesse, Software zu entwickeln, die von jedermann, unabhängig vom benutzten Betriebssystem, benutzt werden kann. Java trug in großem Maße dazu bei, echt plattformunabhängige Software-Entwicklung Wirklichkeit werden zu lassen. Allerdings hatte Java bis vor kurzem keine Antwort auf das Problem der Komponenten-Software - dazu kommen wir gleich.

Als ob das Problem der Plattformabhängigkeit noch nicht genug ist, leiden einige verfügbare Komponententechnologien darunter, in einer spezifischen Programmiersprache oder für eine spezielle Umgebung entwickelt werden zu müssen. So wie die Plattformabhängigkeit Komponenten zur Laufzeit verkrüppelt, führt Limitierung der Komponentenentwicklung auf eine spezifische Programmiersprache oder Entwicklungsumgebung gleichermaßen zur Verkrüppelung der Komponenten im Entwicklungsbereich. Software-Entwickler würden lieber selbst entscheiden, welche Sprache für eine bestimmte Aufgabe und welche Entwicklungsumgebung die am besten geeignete ist, als gezwungen zu sein, eine den Einschränkungen einer Komponententechnologie unterliegende zu verwenden. Demnach muß sich jede realistische, auf lange Sicht ausgerichtete Komponententechnologie dem Problem der Plattform- und Sprachabhängigkeit stellen. Und das bringt mich zu unserem Thema: JavaBeans. Die JavaBeans-Technologie von JavaSoft ist eine Komponententechnologie, die direkte Antworten auf beide Probleme bietet. Die JavaBeans-Technologie verspricht das Paradigma der Zusammenfügung von Komponenten-Software auf eine neue Ebene zu heben. Im Augenblick befindet sich die JavaBeans-Spezifikation in der Entwicklung und die vorläufige Freigabe soll im Anschluß daran erfolgen.

JavaBeans wird als architektur- und plattformunabhängiges API zur Erstellung und Verwendung dynamischer Java-Software-Komponenten realisiert. JavaBeans setzt an dem Punkt an, wo andere Komponententechnologien aufgehört haben, und verwendet die portable Java-Plattform als Basis zur Bereitstellung einer kompletten Lösung für Komponenten-Software, die in der Welt des Online ohne weiteres anwendbar ist.

Das Ziel von JavaBeans

Nach dem schnellen Erfolg des Laufzeitsystems und der Programmiersprache von Java, war JavaSoft die Bedeutung der Entwicklung einer kompletten Lösung für die Komponententechnologie klar. Man antwortete mit der JavaBeans-Technologie, deren Designziele in folgender Spezifikationsliste zusammengefaßt werden können:

- Kompakt, leicht zu erstellen und zu benutzen
- Völlig portabel
- Basiert auf der inhärenten Stärke von Java

- Robuster, verteilter Rechenmechanismus
- Unterstützung für flexible Entwicklungszeit der Komponenteneditoren

Die erste Anforderung von JavaBeans - so kompakt zu sein - beruht auf der Tatsache, daß die JavaBeans-Komponenten häufig in verteilten Umgebungen verwendet werden, in denen komplette Komponenten eventuell über eine Internet-Verbindung mit geringer Bandbreite übertragen werden. Genauer gesagt müssen Komponenten so kompakt wie möglich sein, um eine angemessene Übertragungszeit zu ermöglichen. Der zweite Teil dieses Designziels bezieht sich auf die Leichtigkeit, mit der diese Komponenten erstellt und verwendet werden. Leicht verwendbare Komponenten kann man sich relativ leicht vorstellen, aber die Erzeugung einer Komponentenarchitektur, die das Erstellen von Komponenten leicht gestaltet, ist eine völlig andere Sache. Versuche auf der Ebene der Komponenten-Software werden dem Entwickler oftmals durch komplexes Programmieren von APIs erschwert. JavaBeans-Komponenten müssen also nicht nur leicht zu verwenden, sondern auch leicht zu entwickeln sein. Dies stellt für Sie und mich eine wesentliche Anforderung dar, da sie weniger Kopfschmerzen bereitet und uns mehr Zeit gibt, die Komponenten mit blumigen Eigenschaften auszuschnücken.

JavaBeans-Komponenten basieren größtenteils auf der bereits in der traditionellen Applet-Programmierung von Java verwendeten Klassenstruktur, was wiederum für diejenigen von uns, die viel Zeit und Energie darauf aufwenden, Java zu erlernen, einen enormen Vorteil bietet. JavaSoft hat versprochen, daß die rund um das AWT-Paket entworfenen Java-Applets sich leicht in neue JavaBeans-Komponenten verwandeln lassen. Auch dies leistet den positiven Nebeneffekt der ausgesprochenen Kompaktheit von JavaBeans-Komponenten, da Java-Applets hinsichtlich der Größe bereits sehr leistungsfähig sind.

Das zweite Hauptziel von JavaBeans ist vollständige Portabilität; darüber haben wir zu Beginn dieser Lektion gesprochen. Als Folge davon brauchen sich Entwickler keine Gedanken um die Einbeziehung plattformspezifischer Bibliotheken bei Ihren Java-Applets zu machen. Das Ergebnis sind wiederverwendbare Komponenten, die die Computerwelt glücklich und friedlich unter einem Dach vereinigen. (Okay, vielleicht ist das etwas zu viel verlangt - ich begnüge mich schon mit der Entwicklung einer Komponente, die ohne Anbringen von Modifizierungen auf einem Java-unterstützten System läuft.)

Die bestehende Java-Architektur bietet bereits eine große Anzahl von Vorteilen, die leicht auf Komponenten anwendbar ist. Eine der wesentlicheren Eigenschaften von Java ist ihr eingebauter Klassenerkennungsmechanismus, der dynamische Interaktionen zwischen Objekten ermöglicht. Daraus entsteht ein System, in dem sich Objekte unabhängig von ihrem Ursprung oder ihrer Entwicklungsgeschichte ineinander einbauen lassen. Dieser Mechanismus zur Klassenauffindung stellt nicht nur eine nette Eigenschaft von Java dar, sondern ist in jeder Komponentenarchitektur eine notwendige Voraussetzung und für JavaBeans ein Glücksfall, daß diese Funktionalität von Java bereits kostenlos geboten wird. Andere Komponentenarchitekturen mußten vertrackte Registriermechanismen implementieren, um zu dem gleichen Ergebnis zu kommen.

Darüber hinaus erhält JavaBeans aus der bestehenden Java-Funktionalität noch die Persistenz, d.h. die Fähigkeit eines Objektes, seinen internen Status zu speichern und wieder zu laden. Diese Persistenz erfolgt in JavaBeans automatisch einfach durch die Verwendung des in Java bereits vorhandenen Serialisations-Mechanismus. Falls erforderlich, können Entwickler alternativ speziell angepaßte Persistenz-Lösungen erstellen.



Persistenz ist die Fähigkeit eines Objektes, seinen internen Status zu speichern und wieder zu laden. *Serialisation* ist das Verfahren, Informationen über ein Standardprotokoll zu speichern oder zu laden.

Obwohl kein Schlüsselement der JavaBeans-Architektur, stellt die Unterstützung für verteiltes Rechnen bei JavaBeans dennoch ein Hauptthema dar. Da verteiltes Rechnen, als ein Ergebnis der komplexen Natur der verteilten Systeme, relativ komplexe Lösungen erforderlich macht, hebt JavaBeans die Verwendung von externen verteilten, auf Bedarf basierten Methoden an. Anders ausgedrückt ermöglichen JavaBeans Entwicklern die Verwendung verteilter Rechenmechanismen, wann immer erforderlich, überladen sich aber auch nicht mit entsprechender Unterstützung für verteiltes Rechnen. Man könnte jetzt die JavaBeans-Architekten für faul halten, Tatsache ist aber, daß genau in diesem Designansatz der Schlüssel für die Kompaktheit der

JavaBeans-Komponenten liegt, da verteilte Rechenlösungen unvermeidlich zu höherer Systemverwaltungszeit führen.

Entwickler von JavaBeans-Komponenten haben die Möglichkeit, die für ihre Bedürfnisse am besten geeignete Methode auszuwählen. Mit seiner Technologie der Remote Method Invocation (RMI, Methoden-Fernaufwurf) bietet JavaSoft eine verteilte Rechenlösung, die aber den JavaBeans-Entwicklern keinesfalls die Hände bindet. Unter anderem beinhalten andere Lösungsmöglichkeiten CORBA (Common Object Request Broker Architecture) und von Microsoft DCOM (Distributed Component Object Model). Der Punkt ist, daß verteiltes Rechnen sauberlich von JavaBeans abgetrennt wurde, um die Struktur straff zu halten und gleichzeitig Entwicklern, die die entsprechende Unterstützung benötigen, einen breiten Bereich an Optionen zu bieten. Das letzte Designziel von JavaBeans behandelt Probleme der Entwurfszeit und die Art und Weise, wie Entwickler Anwendungen unter Verwendung von JavaBeans-Komponenten erstellen. Die Architektur von JavaBeans umfaßt Unterstützung für die Spezifizierung von Eigenschaften der Entwurfszeit und Bearbeitungsmechanismen zur weiteren Erleichterung visueller Bearbeitung von JavaBeans-Komponenten. Das hat zur Folge, daß Entwickler visuelle Werkzeuge zum nahtlosen Zusammenfügen und Modifizieren von JavaBeans-Komponenten einsetzen können, ganz ähnlich der Weise, in der bestehende visuelle PC-Werkzeuge mit Komponenten wie VBX- oder OCX-Steuerungen arbeiten. Komponentenentwickler spezifizieren so die Art und Weise, in der die Komponenten in einer Entwicklungsumgebung zu verwenden und zu manipulieren sind. Allein diese Eigenschaft wird offiziell die Verwendung von professionellen visuellen Editoren einleiten und der Produktivität von Anwendungsentwicklern in bedeutender Weise Auftrieb verleihen.

Wie JavaBeans in Beziehung zu Java steht

Für viele Entwickler, die nicht ganz mit dem Gedanken von Software-Komponenten vertraut sind, wird die Beziehung zwischen JavaBeans und Java etwas verwirrend sein. Wurde für Java nicht als eine objektorientierte Technologie mit der Fähigkeit, wiederverwendbare Objekte zu bedienen, die Werbetrommel gerührt? Ja und Nein. Ja, Java bietet eine Möglichkeit der Erstellung wiederverwendbarer Objekte, allerdings gibt es einige Regeln oder Standards, die für die Art und Weise, wie Objekte miteinander interagieren, maßgeblich sind. Durch Spezifizieren umfangreicher Gruppen von Mechanismen für die Interaktion zwischen Objekten, zusammen mit allgemeinen, von den meisten Objekten zu unterstützenden Aktionen wie Persistenz- und Ereignisbehandlung, baut JavaBeans auf dem bestehenden Design von Java auf.

Das aktuelle Java-Komponentenmodell ist, obgleich es nicht schlecht ist, relativ limitiert, wenn es darum geht, echte Wiederverwendung und Interoperabilität zu liefern. Auf der Objektebene gibt es definitiv keinen einfachen Mechanismus zur Erstellung wiederverwendbarer Java-Objekte, die mit anderen Objekten dynamisch in konsequenter Weise in Wechselwirkung treten können. Was Sie dem am nächsten kommend in Java durchführen können, ist, Applets zu erstellen und zu versuchen, diese auf einer Webseite miteinander kommunizieren zu lassen, was nicht gerade eine leichte Aufgabe ist. JavaBeans stellt den Rahmen, in dem diese Kommunikation stattfinden kann, mit Leichtigkeit zur Verfügung. Noch wichtiger ist die Tatsache, daß JavaBeans-Komponenten leicht über eine Standardgruppe von gut definierten Eigenschaften getriggert werden können. JavaBeans vereinigt die Leistungsstärke eines voll ausgereiften Java-Applet mit der Kompaktheit und Wiederverwendbarkeit von Java-AWT-Komponenten, wie beispielsweise Schaltflächen.

JavaBeans-Komponenten sind jedoch nicht auf visuelle Objekte wie Schaltflächen beschränkt. Sie können genauso einfach nicht visuelle JavaBeans-Komponenten entwickeln, die zusammen mit anderen Komponenten einige Hintergrundfunktionen ausüben. Auf diese Weise vereinigt JavaBeans die Leistungsstärke visueller Java-Applets mit nicht visuellen Java-Anwendungen unter dem festen Dach eines Komponentengerüsts.



Eine *nicht visuelle Komponente* ist jede Komponente, deren Ausgabe nicht sichtbar ist. Wenn man sich eine Komponente im Hinblick auf AWT-Objekte wie Schaltflächen und Menüs betrachtet, mag das ein wenig fremd erscheinen. Vergessen Sie dabei aber nicht, daß eine Komponente einfach ein dicht gepacktes Programm ist und nicht unbedingt visuell sein muß. Ein gutes Beispiel für eine nicht visuelle Komponente ist eine

Timer-Komponente, die in bestimmten Abständen Ereignisse sendet. Timer-Komponenten sind in anderen Entwicklungsumgebungen von Komponenten wie Microsoft Visual Basic sehr populär.

Sie können eine Vielzahl von JavaBeans-Komponenten gemeinsam benutzen, ohne daß Sie unter Verwendung visueller Werkzeuge einen Code schreiben müssen. Diese Möglichkeit der gleichzeitigen Benutzung einer Vielzahl von Komponenten, deren Ursprung dabei unwichtig ist, stellt eine Verbesserung des aktuellen Java-Modells dar. Sicherlich können Sie andere, in Java bereits integrierte Objekte benutzen, müssen aber über ausführliches Wissen, die Schnittstelle des Objektes betreffend, verfügen. Darüber hinaus müssen Sie das Objekt programmatisch in Ihren Code integrieren. JavaBeans-Komponenten legen ihre eigenen Schnittstellen visuell dar und stellen somit ein Mittel zur Bearbeitung ihrer Eigenschaften ohne Programmierung bereit. Des weiteren können Sie mit der Verwendung eines visuellen Editors eine JavaBeans-Komponente einfach in eine Anwendung einfügen, ohne daß Sie eine einzige Zeile Code schreiben müssen. Hier sehen wir eine vollständig neue Ebene der Flexibilität und Wiederverwendung, die mit Java allein bisher nicht möglich war.

Das JavaBeans-API

Okay, genug gesagt über JavaBeans und darüber, was es kann und warum es »cool« ist. Konzentrieren wir uns jetzt auf einige Einzelheiten, um festzustellen, wie das alles möglich ist. Behalten Sie dabei im Auge, daß JavaBeans letztendlich eine Programmierschnittstelle ist, was bedeutet, daß all seine Eigenschaften als Erweiterungen der Standard-Klassenbibliothek von Java realisiert werden. Somit wird die gesamte, von JavaBeans zur Verfügung gestellte Funktionalität tatsächlich in dem JavaBeans-API realisiert. Das JavaBeans-API selbst ist eine Reihe von kleineren APIs, die spezifischen Funktionen oder Services gewidmet sind. Die nachfolgende Liste zeigt wesentliche Komponentenservices in dem JavaBeans-API, die zur Erleichterung all der Eigenschaften, die Sie heute gelernt haben, notwendig sind:

- Vereinigung der grafischen Benutzeroberfläche
- Persistenz
- Ereignisbehandlung
- Introspektion
- Application Builder Support

Durch entsprechendes Verständnis dieser Services und wie sie arbeiten, bekommen Sie einen größeren Einblick in die Technologie von JavaBeans. Jeder dieser Services wird in Form von kleineren, in dem größeren API enthaltenen APIs realisiert. In den nächsten Abschnitten widmen wir uns jedem dieser APIs und erklären, warum sie notwendige Elemente der JavaBeans-Architektur darstellen.

Die die grafische Benutzeroberfläche vereinigenden APIs bieten einer Komponente ein Werkzeug zur Vereinigung ihrer Elemente der grafischen Benutzeroberfläche mit dem Container-Dokument, das normalerweise nur die die Komponente beinhaltende Webseite ist. Die meisten Container-Dokumente haben Menüs und Werkzeugleisten, die zur Anzeige der speziellen, von dieser Komponenten bereitgestellten Eigenschaften dienen. Die die grafische Benutzeroberfläche vereinigenden APIs ermöglichen der Komponente, dem Menü und der Werkzeugleiste des Container-Dokuments Eigenschaften hinzuzufügen. Diese APIs definieren auch den Mechanismus, der Raumverhandlungen zwischen den Komponenten und ihren Containern ermöglicht. Anders gesagt, die die grafische Benutzeroberfläche vereinigenden APIs sind auch für die Definition der Layout-Eigenschaften von Komponenten zuständig.

Ein Container-Dokument ist ein JavaBeans-Komponenten enthaltendes Dokument (normalerweise HTML), das als Eltern für alle in ihm enthaltenen Komponenten dient. Neben anderen Dingen sind Container-Dokumente normalerweise für die Verwaltung des Hauptmenüs und der Werkzeugleiste zuständig.

Die Persistenz-APIs spezifizieren den Mechanismus, mit dem Komponenten innerhalb des Kontexts eines Container-Dokumentes gespeichert und geladen werden können. Komponenten erben per Voreinstellung den automatischen, von Java bereitgestellten Serialisations-Mechanismus. Entwickler haben ebenfalls die Freiheit, auf den speziellen Erfordernissen ihrer Komponenten basierende, besser ausgearbeitete Persistenz- Lösungen zu entwerfen.

Die APIs zur Ereignisbehandlung spezifizieren eine ereignisgesteuerte Architektur, die die Wechselwirkung der

Komponenten miteinander definiert. Das Java-AWT beinhaltet bereits ein leistungsstarkes Ereignisbehandlungsmodell, das als Grundlage für die Komponenten-APIs zur Ereignisbehandlung dient. Diese APIs sind wesentlich, wenn es darum geht, den Komponenten die Freiheit zu gewähren, in konsequenter Weise miteinander zu interagieren.

Die Introspektions-APIs definieren Techniken, die Komponenten dazu veranlassen, ihre interne Struktur zur Entwurfszeit direkt zur Verfügung zu stellen. Diese APIs enthalten die notwendige Funktionalität, es Entwicklungstools zu ermöglichen, eine Komponente nach ihrem internen Status abzufragen, einschließlich der Schnittstellen, Methoden und Member-Variablen, aus denen die Komponente besteht. Die APIs sind, basierend auf der Ebene, auf der sie benutzt werden, in zwei getrennte Abschnitte unterteilt. Die Introspektions-APIs der unteren Ebene ermöglichen beispielsweise Entwicklungswerkzeugen direkten Zugriff auf die Komponenteneigenschaften, also eine Funktion, die Sie nicht unbedingt in den Händen von Komponentenbenutzern sehen wollen. APIs der höheren Ebene verwenden APIs der unteren Ebene zur Bestimmung der Teile einer Komponente, die zur Änderung durch den Benutzer exportiert werden. Das heißt, obwohl Entwicklungswerkzeuge zweifellos Gebrauch von beiden Arten von APIs machen, werden sie die APIs der höheren Ebene nur dann verwenden, wenn sie dem Benutzer Komponenteninformationen bereitstellen.

Application Builder Support APIs stellen die bei Entwurfszeit für Bearbeitung und Manipulation der Komponenten erforderliche Systemverwaltungszeit bereit. Diese APIs werden größtenteils von visuellen Entwicklungswerkzeugen eingesetzt, um die Möglichkeit eines visuellen Layouts und Bearbeitung von Komponenten während der Erstellung einer Anwendung zu bieten. Der Teil einer Komponente, der die visuellen Bearbeitungsmöglichkeiten bereitstellt, wurde speziell so entworfen, daß er physisch von der Komponente selbst getrennt ist und somit dazu beiträgt, autonome Laufzeitkomponenten so kompakt wie möglich zu gestalten. In einer reinen Laufzeitumgebung werden Komponenten nur mit der erforderlichen Laufzeitkomponente übertragen. Entwickler, die sich der Vorteile von Entwurfszeitkomponenten bedienen wollen, können leicht den Entwurfszeitteil der Komponente erlangen.

Reicht das vorerst? Oder wollen Sie noch mehr wissen? Die JavaBeans-Spezifikationen sind verfügbar auf der Java-Webseite unter

<http://www.javasoft.com/products/jdk/1.2/docs/guide/beans/index.html>

JavaBeans - Technik, Konzepte, Beispiele von Michael Morrison (Markt & Technik, ISBN 3-8272-5284-9) ist sehr gut als Einführung in JavaBeans und die Möglichkeiten, die es Ihnen bietet, geeignet.

Applet-Tricks

Diese Lektion beginne ich mit ein paar kleinen Tips, die sonst nirgendwo hineinpassen: Die Verwendung von `showStatus()`, um Meldungen in der Browser-Statuszeile auszugeben, Applet-Informationen bereitzustellen und zwischen mehreren Applets auf der gleichen Seite eine Kommunikation zu ermöglichen.

Die `showStatus()`-Methode

Die in der Klasse Applet verfügbare `showStatus()`-Methode ermöglicht Ihnen die Anzeige einer Zeichenkette in der Statuszeile des Browsers, in dem das Applet ausgegeben wird. Sie können dies zum Ausgeben von Fehler-, Verknüpfungs-, Hilfs- oder anderen Statusmeldungen benutzen:

```
getAppletContext().showStatus("Change the color");
```

Die Methode `getAppletContext()` ermöglicht Ihrem Applet Zugriff auf Funktionen des Browsers, in dem es enthalten ist. Sie haben dies teilweise schon im Zusammenhang mit Verknüpfungen kennengelernt, für die Sie die `showDocument()`-Methode benutzt haben, um den Browser zum Laden einer Seite aufzufordern. `showStatus()` bedient sich desselben Mechanismus, um Statusmeldungen auszugeben.



Möglicherweise wird `showStatus()` nicht von allen Browsern unterstützt, deshalb sollten Sie sich in bezug auf die volle Funktionalität oder Schnittstelle Ihres Applet nicht auf den Browser verlassen. Diese Methode bietet eine nützliche Art der Kommunikation mit dem Benutzer - wenn Sie jedoch eine zuverlässigere Methode benötigen, richten Sie in Ihrem Applet ein Label ein und aktualisieren es auf die Meldungsänderungen hin.

Applet-Informationen

Das Abstract Windowing Toolkit bietet einen Mechanismus zum Einbinden von Informationen in Ihr Applet. Normalerweise umfaßt der Browser, in dem das Applet betrachtet wird, eine Möglichkeit zur Anzeige von Bildschirminformationen. Mit diesem Mechanismus können Sie Ihr Applet mit Ihrem Namen oder Ihrer Organisation unterzeichnen oder entsprechende Informationen zur Kontaktaufnahme bereitstellen, damit die Benutzer auf Wunsch mit Ihnen in Verbindung treten können.

Zur Angabe von Informationen über Ihr Applet überschreiben Sie die Methode `getAppletInfo()` :

```
public String getAppletInfo() {
    return "GetRaven copyright 1995 Laura Lemay";
}
```

Verknüpfungen in Applets erstellen

Da Applets auf Browser-Webseiten laufen, stellt die Möglichkeit, die Fähigkeit des Browsers zum Laden neuer Webseiten zu nutzen, ein gute Lösung dar. Java bietet einen Mechanismus, der den Browser zum Laden einer neuen Seite veranlaßt. Sie können diesen Mechanismus beispielsweise zu Erstellung von animierten Imagemaps benutzen, die beim Anklicken eine neue Seite laden.

Zur Verknüpfung einer neuen Seite erstellen Sie eine neue Instanz der URL-Klasse. Sie haben dies bereits im Zusammenhang mit Bildern gesehen. Hier steigen wir aber etwas tiefer in die Materie ein.

Die URL-Klasse stellt einen »Uniform Resource Locator« dar, einen Zeiger auf eine Datei oder ein Objekt im World Wide Web. Um einen neuen URL zu erzeugen, können Sie einen von vier unterschiedlichen Konstruktoren verwenden:

- `URL(String, String, int, String)` erzeugt ein neues URL-Objekt mit einem bestimmten Protokoll (http, ftp, gopher, file), einem Hostnamen (www.lne.com, ftp.netcom.com), einer Portnummer (80 für http) und einem Datei- oder Pfadnamen.
- `URL(String, String, String)` erzeugt dasselbe wie die obige Form, jedoch ohne Portnummer.
- `URL(URL, String)` erzeugt einen URL mit einem bestimmten Basis- und einem relativen Pfad. Für den Basispfad können Sie `getDocumentBase()` als URL der aktuellen HTML-Datei oder `getCodeBase()` als URL der Applet-Klassendatei in Java verwenden. Der relative Pfad wird dem letzten Verzeichnis in diesen Basis-URLs angefügt (genau wie bei Bildern oder Sound).
- `URL(String)` erzeugt ein URL-Objekt aus einer URL-Zeichenkette (das das Protokoll, den Hostnamen, optionalen Portnamen und Dateinamen enthalten sollte).

Bei der letzten Form (Erstellen einer URL aus einer Zeichenkette) müssen Sie eine eventuell falsch gebildete URL-Adresse berücksichtigen, deshalb umgeben Sie den URL-Konstruktor mit einem `try...catch`-Block:

```
String url = "http://www.yahoo.com/";
try { theURL = new URL(url); }
catch ( MalformedURLException e) {
    System.out.println("Bad URL: " + theURL);
}
```

Das Ermitteln und Erzeugen eines URL-Objektes ist der schwierige Teil. Wenn Sie eines haben, brauchen Sie es nur an den Browser weiterzugeben. Das geschieht in einer einzigen Codezeile, wobei die URL das URL-Objekt ist, mit dem die Verknüpfung hergestellt werden soll:

```
getAppletContext().showDocument(theURL);
```


Der Browser, der das Java-Applet mit diesem Code enthält, wird dann das Dokument unter dieser URL laden und anzeigen.

Listing 19.1 beinhaltet die beiden Klassen: ButtonLink und seine Handler-Klasse Bookmark. ButtonLink ist ein einfaches Applet, das drei Schaltflächen anzeigt, die wichtige Web-Standorte darstellen (die Schaltflächen sind in Abbildung 19.1 dargestellt). Durch Anklicken der Schaltflächen wird das Dokument von den Standorten, auf die sich diese Schaltflächen beziehen, geladen.



Abbildung 19.1:
Das Bookmark-Applet



Dieses Applet muß in einem Browser ausgeführt werden, damit die Links funktionieren. Außerdem wurde es mit den Techniken zur Ereignisbehandlung von Java 1.01 geschrieben, um die größtmögliche Zahl von Browsern zu unterstützen. Sie erhalten eine deprecated-Warnung, wenn Sie das Applet mit dem javac-Tool von Java 1.2 kompilieren.

Listing 19.1: Der gesamte Quelltext von Buttonlink.java

```

1: import java.awt.*;
2: import java.net.*;
3:
4: public class ButtonLink extends java.applet.Applet {
5:     Bookmark bmList[] = new Bookmark[3];
6:
7:     public void init() {
8:         bmList[0] = new Bookmark("Teach Yourself Java 1.2 in 21 Days",
9:             "http://www.prefect.com/java21");
10:        bmList[1] = new Bookmark("Pearson Technology
Group Computer Publishing",
11:            "http://www.mcp.com");
12:        bmList[2] = new Bookmark("JavaSoft",
13:            "http://java.sun.com");
14:
15:        GridLayout gl = new GridLayout(bmList.length, 1, 10, 10);
16:        setLayout(gl);
17:        for (int i = 0; i < bmList.length; i++) {

```

```

18:         add(new Button(bmList[i].name));
19:     }
20: }
21:
22: public boolean action(Event evt, Object arg) {
23:     if (evt.target instanceof Button) {
24:         linkTo( (String)arg );
25:         return true;
26:     }
27:     else return false;
28: }
29:
30: void linkTo(String name) {
31:     URL theURL = null;
32:     for (int i = 0; i < bmList.length; i++) {
33:         if (name.equals(bmList[i].name))
34:             theURL = bmList[i].url;
35:     }
36:     if (theURL != null)
37:         getAppletContext().showDocument(theURL);
38: }
39: }
40:
41: class Bookmark {
42:     String name;
43:     URL url;
44:
45:     Bookmark(String name, String theURL) {
46:         this.name = name;
47:         try {
48:             this.url = new URL(theURL);
49:         } catch (MalformedURLException e) {
50:             System.out.println("Bad URL: " + theURL);
51:         }
52:     }
53: }

```

Mit dem folgenden HTML-Code können Sie das Applet in eine Webseite einbinden:

```

<APPLET CODE="ButtonLink.class" HEIGHT=120 WIDTH=240>
</APPLET>

```

Dieses Applet besteht aus zwei Klassen: Die erste, ButtonLink, implementiert das Applet; die zweite, Bookmark, ist eine Klasse, die ein Bookmark (Lesezeichen) darstellt. Bookmarks bestehen wiederum aus zwei Teilen: einem Namen und einer URL.

Dieses spezielle Applet erstellt drei Bookmark-Instanzen (Zeilen 8 bis 13) und speichert sie in einem Bookmark-Array (dieses Applet könnte leicht so abgeändert werden, daß es Bookmarks als Parameter aus einer HTML-Datei enthält). Für jedes Bookmark wird eine Schaltfläche erstellt, deren Beschriftung dem Wert des Bookmark-Namens entspricht.

Durch Anklicken einer Schaltfläche wird die linkTo()-Methode aufgerufen. linkTo() ist in den Zeilen 30 bis 38 definiert und benutzt den Namen der Schaltfläche, die das Ereignis ausgelöst hat, um den eigentlichen URL aus dem Bookmark-Objekt zu ermitteln, und weist dann den Browser an, den URL zu laden, auf den dieses Lesezeichen verweist.

Kommunikation zwischen Applets

Zuweilen möchten Sie vielleicht mehrere Applets in eine HTML-Seite stellen. Um dies zu realisieren, können Sie mehrere verschiedene `<APPLET>`-Tags einbinden. Der Browser erstellt dann die Instanz für jedes Applet, das auf der HTML-Seite erscheint.

Was nun, wenn diese Applets miteinander kommunizieren sollen? Oder wenn in einem Applet etwas geändert wird, das die übrigen Applets in irgendeiner Weise betrifft? Die beste Möglichkeit hierfür ist die Verwendung des Applet-Kontexts, um mehrere Applets in eine Seite einzubinden.



Bevor Sie intensives Arbeiten mit der Kommunikation zwischen Applets beginnen, muß ich Sie vorab warnen, daß der in diesem Abschnitt beschriebene Mechanismus bei unterschiedlichen Browsern und unterschiedlichen Java-Anwendungen auch unterschiedlich (und oft unzuverlässig) realisiert wird. Wenn Sie sich auf die Kommunikation zwischen Applets für Ihre Webseiten verlassen müssen, ist umfangreiches Testen dieser Applets mit unterschiedlichen Browsern auf unterschiedlichen Plattformen absolut notwendig.

Der Applet-Kontext ist in einer entsprechend benannten Klasse, der Klasse `AppletContext`, definiert. Um eine Instanz dieser Klasse für Ihr Applet zu holen, verwenden Sie die Methode `getAppletContext()`. Den Umgang mit der `getAppletContext()`-Methode haben Sie bereits in verschiedenen Beispielen gelernt. Sie können diese Methode auch verwenden, um andere Applets in eine Seite zu stellen. Um beispielsweise eine Methode mit dem Namen `sendMessage()` in allen Applets auf einer Seite aufzurufen (einschließlich des jeweils aktuellen Applets), verwenden Sie die `getApplets()`-Methode und eine `for`-Schleife, die etwa folgendermaßen aussieht:

```
for (Enumeration e = getAppletContext().getApplets();
     e.hasMoreElements();) {
    Applet current = (MyAppletSubclass)(e.nextElement());
    current.sendMessage();
}
```

Die `getApplets()`-Methode gibt ein `Enumeration`-Objekt mit einer Liste der auf der Seite befindlichen Applets aus. Die `for`-Schleife für das `Enumeration`-Objekt ermöglicht Ihnen den Zugriff auf alle Elemente nacheinander in der Auflistung. Beachten Sie, daß jedes Element in dem `Enumeration`-Objekt eine Instanz der Objektklasse ist; um ein Ihren Wünschen entsprechendes Verhalten des Applet zu erreichen (und Meldungen von anderen Applets aufzunehmen), müssen Sie es so gestalten, daß es eine Instanz Ihrer Applet-Subklasse ist (in diesem Fall der Klasse `MyAppletSubclass`).

Etwas komplizierter wird es, wenn Sie eine Methode in einem bestimmten Applet aufrufen wollen. Hierfür geben Sie jedem Applet einen Namen und setzen in den Rumpf des betreffenden Applet-Codes eine Referenz auf diesen Namen.

Um einem Applet einen Namen zu geben, benutzen Sie das `NAME`-Attribut für `<APPLET>` in Ihrer HTML-Datei:

```
<P>This applet sends information:
<APPLET CODE="MyApplet.class" WIDTH=100 HEIGHT=150
  NAME="sender"> </APPLET>
<P>This applet receives information from the sender:
<APPLET CODE="MyApplet.class" WIDTH=100 HEIGHT=150
  NAME="receiver"> </APPLET>
```

Um eine Referenz zu einem anderen Applet auf der gleichen Seite zu erhalten, verwenden Sie die `getApplet()`-Methode aus dem Applet-Kontext mit dem Namen des betreffenden Applets. Dadurch erhalten Sie eine Referenz zu dem Applet dieses Namens. Sie können dann auf das Applet so verweisen, als wäre es nur ein anderes Objekt: Methoden aufrufen, seine Instanzvariablen setzen usw. Hier ein entsprechender Code dafür:

```
// Receiver-Applet holen
Applet receiver = (MyAppletSubclass)getAppletContext().getApplet("receiver");
```

```
// Anweisen zur Aktualisierung.
receiver.update(text, value);
```

In diesem Beispiel wird die `getApplet()`-Methode verwendet, um eine Referenz zu dem Applet mit dem Namen `receiver` zu erhalten. Beachten Sie, daß das von `getApplet()` ausgegebene Objekt eine Instanz der generischen Applet-Klasse ist; es ist anzunehmen, daß Sie dieses Objekt einer Instanz Ihrer Subklasse zuweisen wollen. Mit der Referenz zu dem benannten Applet können Sie dann Methoden in diesem Applet aufrufen, als ob es ein anderes Objekt in Ihrer eigenen Umgebung wäre. Sie können beispielsweise, vorausgesetzt beide Applets besitzen eine `update()`-Methode, den Empfänger anweisen, sich selbst unter Verwendung der Informationen des aktuellen Applet zu aktualisieren.

Indem Sie Ihren Applets Namen geben und sich dann auf sie mit den in diesem Abschnitt beschriebenen Methoden beziehen, können Ihre Applets miteinander kommunizieren und sich auf die jeweils geänderten Informationen des anderen Applets aktualisieren, so daß alle Applets auf Ihrer Seite ein einheitliches Verhalten aufweisen.

Ausschneiden, Kopieren und Einfügen

Beginnend mit der Version 1.1 des AWT wurde die Unterstützung für Ausschneiden-, Kopieren- und Einfügen-Operationen zwischen Komponenten einer AWT-Benutzeroberfläche und anderen Programmen, die auf der Plattform ausgeführt werden und nicht in Java implementiert sind, hinzugefügt. Zuvor ermöglichte es das AWT nur, Daten zwischen solchen Komponenten zu kopieren und einzufügen, in denen auf den systemeigenen Plattformen bereits grundlegende Fähigkeiten hierfür integriert waren (Text konnte beispielsweise zwischen Textfeldern und Textbereichen kopiert und eingefügt werden). Mit der Version 1.1 wurde diese Fähigkeit so erweitert, daß auch andere Daten oder Objekte von einer Komponente in eine andere übertragen werden können.

Um Daten von einer Komponente in eine andere übertragen zu können, müssen Sie zuerst ein austauschbares Objekt definieren und dann Komponenten modifizieren oder erstellen, die die Fähigkeit besitzen, dieses Objekt zu übertragen.

Die Klassen und Schnittstellen hierfür sind im `java.awt.datatransfer`-Paket enthalten.

Austauschbare Objekte erstellen

Daten-Flavors werden durch MIME-Typen beschrieben, demselben Mechanismus, der von zahlreichen E-Mail-Programmen und dem World Wide Web selbst angewandt wird, um Inhalte zu kennzeichnen. Wenn Sie mit MIME-Typen nicht vertraut sind, können Sie in RFC 1521 die MIME-Spezifikation nachlesen (es sollte Ihnen möglich sein, sich diese Spezifikation an jedem Web- oder FTP-Standort, der die verschiedenen Internet-RFC-Dokumente enthält, zu beschaffen, <http://ds.internic.net/rfc/rfc1521.txt> ist ein Beispiel hierfür). Zusätzlich zu dem logischen Flavor-Namen, hat das Daten-Flavor ebenfalls einen »benutzerlesbaren« Flavor-Namen, der für unterschiedliche internationale Sprachen entsprechend übersetzt werden kann. Daten-Flavors können außerdem eine entsprechende Java-Klasse haben - wenn beispielsweise das Daten-Flavor eine Unicode-Zeichenkette ist, würde die `String`-Klasse dieses Flavor repräsentieren. Wenn das Flavor keine es repräsentierende Klasse besitzt, wird die Klasse `InputStream` benutzt.



Ein *austauschbares Objekt* ist ein Objekt, das mit dem AWT-Datenübertragungsmechanismus von einer Komponente in eine andere transportiert werden kann und eine bestimmte Menge von zu übertragenden Daten einkapselt (zum Beispiel formatierten Text). Genauer gesagt ist ein austauschbares Objekt ein Objekt, das die Transferable- Schnittstelle implementiert.

Wenn Sie ein austauschbares Objekt erstellen, müssen Sie als erstes entscheiden, welche »Flavors« das Objekt unterstützen soll. Ein *Flavor* ist im wesentlichen das Format der zu übertragenden Daten. Wenn Sie beispielsweise HTML-formatierten Text aus einem Browser kopieren und versuchen, ihn einzufügen, könnten die Daten in einem

von mehreren unterschiedlichen Flavors eingefügt werden: als formatierter Text, als Klartext oder als HTML-Code. Daten-Flavors bestimmen, wie der Teil, von dem kopiert wird, und der Teil, in den eingefügt wird, verhandeln, wie die Daten selbst übertragen werden sollen. Wenn die Quelle und das Ziel der Datenübertragung nicht die gleiche Gruppe von Flavors unterstützen, können die Daten nicht übertragen werden.

Um einen neuen Daten-Flavor zu erstellen, erzeugen Sie eine neue Instanz von der DataFlavor-Klasse, indem Sie einen der beiden folgenden Konstruktoren verwenden:

- DataFlavor(Class, String) erzeugt ein DataFlavor-Objekt, das eine Java-Klasse darstellt. Das String-Argument dient der benutzerlesbaren Flavor-Bezeichnung. Das daraus resultierende DataFlavor-Objekt wird einen MIME-Typ application/ x-javaserIALIZEDobject haben.
- DataFlavor(String, String) erzeugt ein DataFlavor-Objekt, das einen MIME- Typ repräsentiert, wobei das erste Argument der Mime-Typ und das zweite die benutzerlesbare Zeichenkette ist. Die dieses Flavor repräsentierende Klasse ist InputStream .

Mit diesem DataFlavor-Objekt können Sie seine Werte abfragen oder MIME-Typen mit anderen DataFlavor-Objekten vergleichen, um die Übertragungsweise der Daten zu verhandeln.

Daten-Flavors werden von austauschbaren Objekten benutzt, die unter Verwendung der Transferable-Schnittstelle definiert werden. Ein austauschbares Objekt beinhaltet die zu übertragenden Daten und Instanzen jedes Daten-Flavors, das dieses Objekt repräsentiert. Darüber hinaus müssen Sie die Methoden getTransferDataFlavors(), isDataFlavorSupported() und getTransferData() implementieren, damit Ihr übertragbares Objekt auch effektiv verhandelt und übertragen werden kann (Einzelheiten hierzu finden Sie bei der Transferable-Schnittstelle).

Die Klasse StringSelection implementiert ein einfaches übertragbares Objekt zum Übertragen von Zeichenketten und verwendet dabei sowohl DataFlavor-Objekte als auch die Transferable-Schnittstelle. Wenn Sie hauptsächlich Text kopieren möchten, ist StringSelection hierfür die beste Stelle zum Ansatz (und möglicherweise das einzige übertragbare Objekt, das Sie effektiv benötigen). Eingehendes Untersuchen der Quelle der StringSelection-Klasse hilft Ihnen auch herauszufinden, auf welche Weise übertragbare Objekte arbeiten. (Sie können die Quelle mit dem JDK selbst finden, und sie ist ebenfalls in der Datenübertragungsspezifikation unter <http://java.sun.com/products/jdk/1.1/docs/guide/awt/designspec/datatransfer.html>) angegeben.

Beachten Sie, daß austauschbare Objekte lediglich zum Einkapseln von Daten und zur Beschreibung des Formats verwendet werden; sie führen in keinsten Weise Formatierung dieser Daten durch. Dafür ist Ihr Programm zuständig, wenn Sie die Zwischenablage zur Einholung von Daten aus einer Quelle verwenden.

Verwendung der Zwischenablage

Nachdem Sie ein übertragbares Objekt erzeugt haben, können Sie eine Zwischenablage zur Übertragung dieses Objektes zwischen Komponenten und aus Java heraus auf die systemeigene Plattform verwenden. Java 1.2 bietet einen sehr leicht zu handhabenden Mechanismus für die Zwischenablage, der es Ihnen ermöglicht, Daten in die Zwischenablage zu stellen und Daten wieder aus dieser Zwischenablage zu laden. Sie können entweder eine einzelne Standardsystem-Zwischenablage verwenden, um Daten in andere und aus anderen Programmen, die auf der systemeigenen Plattform laufen, zu transportieren, oder Sie können Ihre eigenen Instanzen der Zwischenablage zur Erstellung besonderer Zwischenablagen oder mehrerer Gruppen von Spezialzwischenablagen verwenden.

In Java werden Zwischenablagen durch die Klasse Clipboard repräsentiert, die auch ein Bestandteil des java.awt.datatransfer-Pakets ist. Das Standardsystem Clipboard erhalten Sie durch Verwendung der Methoden getToolkit() und getSystemClipboard() (wie Sie im letzten Abschnitt gelernt haben, bietet getToolkit() eine Möglichkeit, auf verschiedene systemeigene Fähigkeiten zuzugreifen) wie folgt:

```
Clipboard clip = getToolkit().getSystemClipboard()
```

Hier noch ein wichtiger Hinweis in bezug auf die Systemzwischenablage - aus Sicherheitsgründen können Applets derzeit nicht auf die Systemzwischenablage zugreifen (möglicherweise befinden sich sensitive Daten in dieser Zwischenablage). Dies hindert Applets daran, irgendwelche Daten in die oder von der systemeigenen Plattform zu kopieren oder einzufügen (mit Ausnahme der Fähigkeiten, die dort bereits vorhanden sind, wie Text innerhalb von

Textfeldern und Textbereichen). Allerdings haben Sie die Möglichkeit, Ihre eigenen internen Zwischenablagen zum Kopieren und Einfügen zwischen Komponenten in einem Applet zu verwenden.

Jede Komponente, die Gebrauch von der Zwischenablage machen möchte - entweder um Daten unter Verwendung von Ausschneiden oder Kopieren in die Zwischenablage zu stellen, oder um Daten unter Verwendung von Einfügen aus der Zwischenablage zu holen -, muß die ClipboardOwner-Schnittstelle implementieren. Diese Schnittstelle verfügt über eine Methode: `lostOwnership()`, die aufgerufen wird, wenn eine andere Komponente die Kontrolle der Zwischenablage übernimmt.

Zur Durchführung von Ausschneiden oder Kopieren, d.h. um Daten in die Zwischenablage zu stellen, führen Sie folgende Schritte aus:

1. Erzeugen Sie eine Instanz Ihres Transferable-Objekts, um die zu kopierenden Daten zu speichern.
2. Erzeugen Sie eine Instanz des Objekts, das die ClipboardOwner-Schnittstelle realisiert (das kann entweder die aktuelle Klasse sein, oder das Transferable-Objekt kann auch den Inhaber der Zwischenablage realisieren).
3. Wenn Sie die Systemzwischenablage erneut verwenden, benutzen Sie `getSystemClipboard()`, um eine Referenz zu dieser Zwischenablage zu erhalten.
4. Rufen Sie die `setContents()`-Methode der Zwischenablage mit dem Transferable -Objekt und dem Objekt, das ClipboardOwner als Argumente implementiert, auf. Mit dieser Methode erhält Ihr Objekt erklärtes »Eigentumsrecht« an der Zwischenablage.
5. Übernimmt ein anderes Objekt die Zwischenablage, wird die Methode `lostOwnership()` aufgerufen. Ist das der Fall, wollen Sie implementieren, daß diese Methode etwas unternimmt (oder eine leere Methode erzeugen, wenn es Ihnen nichts ausmacht, daß jemand den Inhalt Ihrer Zwischenablage ersetzt hat).

Um die Einfügen-Operation zu implementieren, d.h. Daten aus einer Zwischenablage zu entnehmen, führen Sie die folgenden Schritte aus:

1. Benutzen Sie die `getContents()`-Methode der Zwischenablage, die ein austauschbares Objekt ausgibt.
2. Benutzen Sie die `getTransferDataFlavors()`-Methode des austauschbaren Objekts, um zu untersuchen, welche Flavors dieses austauschbare Objekt unterstützt. Legen Sie fest, welcher Flavor benutzt werden soll.
3. Laden Sie die Daten mit dem richtigen Flavor, indem Sie die Methode `getTransferData()` des austauschbaren Objektes verwenden.

Hier ein sehr einfaches Anwendungsbeispiel, in dem die Zwischenablage zum Kopieren von Text aus einem Textfeld in ein anderes benutzt wird (wie in Abbildung 19.2 gezeigt). Listing 19.2 zeigt den Code für dieses Beispiel auf.

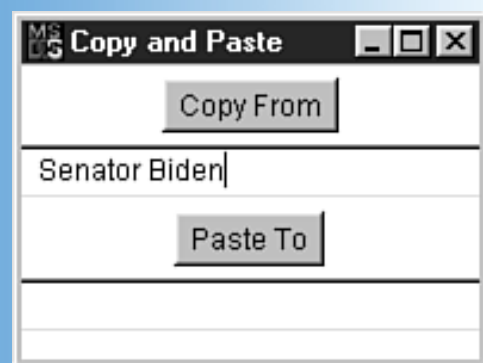


Abbildung 19.2:
Die CopyPaste-Applikation

Listing 19.2: Kopieren und Einfügen

```
1: import java.awt.*;
2: import java.awt.event.*;
```

```
3: import java.awt.datatransfer.*;
4:
5: public class CopyPaste extends Frame
6:     implements ActionListener, ClipboardOwner {
7:
8:     Button copy, paste;
9:     TextField tfCopy, tfPaste;
10:    Clipboard clip;
11:
12:    public static void main(String[] arguments) {
13:        CopyPaste test = new CopyPaste();
14:        WindowListener l = new WindowAdapter() {
15:            public void windowClosing(WindowEvent e) {
16:                System.exit(0);
17:            }
18:        };
19:        test.addWindowListener(l);
20:        test.setSize(200, 150);
21:        test.show();
22:    }
23:
24:    CopyPaste() {
25:        super("Copy and Paste");
26:        clip = getToolkit().getSystemClipboard();
27:        FlowLayout flo = new FlowLayout();
28:        setLayout(flo);
29:
30:        copy = new Button("Copy From");
31:        tfCopy = new TextField(25);
32:        paste = new Button("Paste To");
33:        tfPaste = new TextField(25);
34:
35:        copy.addActionListener(this);
36:        paste.addActionListener(this);
37:        paste.setEnabled(false);
38:
39:        add(copy);
40:        add(tfCopy);
41:        add(paste);
42:        add(tfPaste);
43:    }
44:
45:    void doCopy() {
46:        if (tfCopy.getText() != null) {
47:            String txt = tfCopy.getText();
48:            StringSelection trans = new StringSelection(txt);
49:            clip.setContents(trans, this);
50:            paste.setEnabled(true);
51:        }
52:    }
53:
54:    void doPaste() {
55:        Transferable toPaste = clip.getContents(this);
56:        if (toPaste != null) {
57:            try {
```

```

58:         String txt = (String)toPaste.getTransferData(
59:             DataFlavor.stringFlavor);
60:         tfPaste.setText(txt);
61:         paste.setEnabled(false);
62:     } catch (Exception e) {
63:         System.out.println("Error -- " + e.toString());
64:     }
65: }
66: }
67:
68: public void actionPerformed(ActionEvent e) {
69:     if (e.getSource() == copy)
70:         doCopy();
71:     else if (e.getSource() == paste)
72:         doPaste();
73: }
74:
75: public void lostOwnership(Clipboard clip,
76:     Transferable contents) {
77: }
78: }

```

Der meiste Code in dieser Applikation beinhaltet Verhaltensweisen, um eine grafische Benutzeroberfläche zu erstellen und auf Benutzer-Ereignisse zu reagieren. Sie lernen während der nächsten zwei Tage, wenn Sie Swing-Applikationen mit Java 1.2 entwickeln, mehr über diese Techniken.

Folgende Anweisungen beziehen sich auf das Kopieren und Einfügen::

- In Zeile 3 macht die import-Anweisung das Paket `java.awt.datatransfer` verfügbar.
- In den Zeilen 5 und 6 weist die `implements`-Anweisung darauf hin, daß diese Klasse die Schnittstelle `ClipboardOwner` implementiert.
- In Zeile 10 wird eine Referenz auf ein `Clipboard`-Objekt namens `clip` erzeugt.
- Zeile 26 assoziiert das `Clipboard`-Objekt `clip` über einen Aufruf der Methode `getSystemClipboard()` mit der Systemzwischenablage.
- In den Zeilen 45 bis 52 wird die Methode `doCopy()` aufgerufen, wenn ein Benutzer auf die Schaltfläche `Copy From` klickt. Wenn sich etwas in dem Textfeld `Copy From` befindet, wird in Zeile 47 der Text aus dem Feld extrahiert und ein neues austauschbares Objekt in Zeile 48 erzeugt und in Zeile 49 in der Zwischenablage platziert. Anschließend wird die `Paste`-Schaltfläche aktiviert.
- Die Zeilen 54 bis 66: Die Methode `doPaste()` wird aufgerufen, wenn die Schaltfläche `Paste To` angeklickt wird. Es wird ein `Transferable`-Objekt verwendet, um den Inhalt der Zwischenablage zu prüfen. Falls diese nicht leer ist, werden die Daten übernommen (Zeilen 58 und 59) und in dem Textfeld `Paste To` (Zeile 60) platziert. Sobald dies erledigt ist, wird die `Paste-To`-Schaltfläche deaktiviert.
- In den Zeilen 75-77 wird die Methode `lostOwnership()` implementiert. Diese ist notwendig, da die Klasse die Schnittstelle `ClipboardOwner` implementiert. Allerdings wird in dieser Applikation nichts unternommen, wenn sich die Besitzverhältnisse der Zwischenablage ändern.

RMI (Remote Method Invocation)

RMI wird zur Erstellung von Java-Anwendungen benutzt, die mit anderen Java-Anwendungen über ein Netz kommunizieren können. Um es genauer auszudrücken, ermöglicht RMI einer Java-Anwendung, Methoden und Zugriffsvariablen innerhalb anderer Java-Anwendungen, die in unterschiedlichen Java-Umgebungen oder auf unterschiedlichen Systemen laufen können, aufzurufen und über die Netzverbindung Objekte weiter- und zurückzugeben. RMI ist ein höher entwickelter Mechanismus zur Kommunikation zwischen verteilten Java-Objekten als eine einfache Socket-Verbindung es sein könnte, da der Mechanismus und die Protokolle, durch die Sie zwischen den Objekten kommunizieren, definiert und standardisiert sind. Sie können mit einem anderen

Java- Programm mit RMI kommunizieren, ohne vorher das Protokoll zu kennen.



Eine andere Form der Kommunikation zwischen Objekten nennt sich RPC (Remote Procedure Calls), mit der Sie über eine Netzverbindung in anderen Programmen Methoden aufrufen oder Verfahren ausführen können. Obwohl RPC und RMI viele Gemeinsamkeiten haben, besteht der Hauptunterschied darin, daß RPC nur Verfahrensaufrufe über die Leitung sendet, wobei die Argumente entweder derart weitergeleitet oder beschrieben werden, daß sie auf jeder Seite rekonstruierbar sind. Tatsächlich gibt RMI ganze Objekte über das Netz hin und her und eignet sich daher besser für ein vollkommen objektorientiertes verteiltes Objektmodell.

Das RMI-Konzept könnte Visionen von auf der ganzen Welt fröhlich miteinander kommunizierenden Objekten aufkommen lassen, normalerweise wird RMI jedoch in einer traditionelleren Client-Server-Situation benutzt: Eine Einzel-Server-Anwendung erhält Verbindungen und Anfragen von einer Anzahl Clients. RMI ist einfach ein Mechanismus, der es Client und Server ermöglicht, miteinander zu kommunizieren.

Die RMI-Architektur

Die für RMI gesteckten Ziele waren, ein verteiltes Objektmodell in Java zu integrieren, ohne dabei die Sprache oder das bestehende Objektmodell auseinanderzureißen und das Eintreten in eine Wechselbeziehung mit einem Remote-Objekt so einfach wie mit einem lokalen zu gestalten. Beispielsweise sollte es Ihnen möglich sein, Remote-Objekte auf exakt dieselbe Weise wie lokale zu verwenden (sie Variablen zuweisen, sie als Argumente an Methoden weitergeben usw.), und Remote-Objekte in Methoden aufzurufen, sollte auf dieselbe Weise erreichbar sein, wie für entsprechende lokale Aufrufe. Darüber hinaus beinhaltete RMI einen höher entwickelten Mechanismus, um Methoden bei Remote-Objekten aufzurufen, um ganze Objekte oder Teile von Objekten entweder mit Referenz oder mit Wert weiterzugeben sowie zusätzliche Ausnahmen für die Bearbeitung von Netzfehlern, die beim Durchführen von Remote-Operationen vorkommen können.

Zum Erreichen dieser Ziele besitzt RMI mehrere Ebenen und ein einzelner Methodenaufruf durchläuft viele dieser Ebenen, um an seinen Bestimmungsort zu gelangen (siehe Abbildung 19.3). Es gibt drei Ebenen:

- Die Ebenen »Stubs« und »Skeletons« auf dem Client bzw. Server. Diese Ebenen agieren auf beiden Seiten als Stellvertreter-Objekte, indem Sie die »Remoteness« des Methodenaufrufs vor der eigentlichen Implementierungsklasse verbergen. Sie können also in Ihrer Client-Anwendung Remote-Methoden auf exakt dieselbe Art und Weise wie lokale Methoden aufrufen; das Stub-Objekt ist ein lokaler Stellvertreter für das Remote-Objekt.
- Die Ebene »Remote Reference Layer« ist für die Handhabung der Zusammenstellung eines Methodenaufrufs und seiner Parameter und Rückgabewerte für den Transport über das Netz zuständig.
- Die Transportebene, die eigentliche Netzverbindung von einem System zu einem anderen.

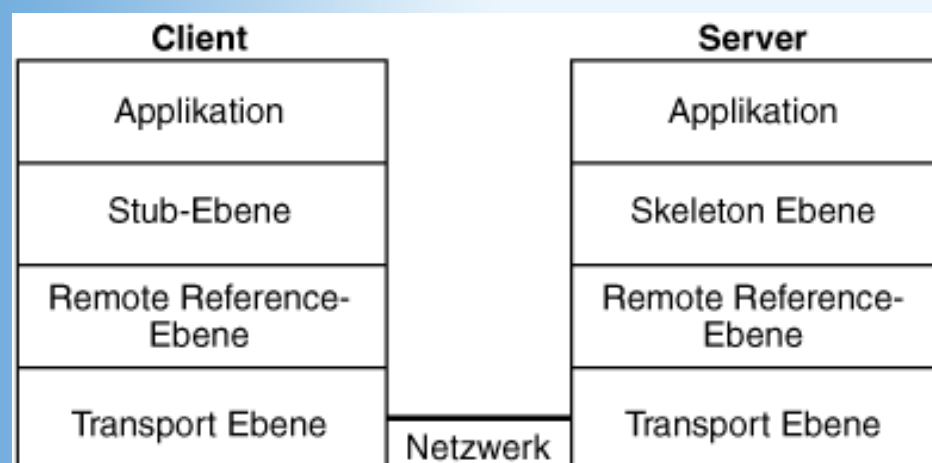


Abbildung 19.3:

RMI-Ebenen

Die Verfügbarkeit von drei RMI-Ebenen ermöglicht die unabhängige Kontrolle und Implementierung jeder Ebene. Stubs und Skeletons ermöglichen es Client- und Server- Klassen, sich so zu verhalten, als ob die vorliegenden Objekte lokale seien, und genau dieselben Java-Spracheigenschaften zum Zugriff auf diese Objekte zu verwenden. Die Ebene Remote Reference Layer stellt die Verarbeitung des Remote-Objekts in seine eigene Ebene, die dann unabhängig von den Anwendungen, die von ihr abhängig sind, optimiert und reimplementiert werden kann. Schließlich wird die Ebene Network Transport Layer unabhängig von den beiden anderen benutzt, so daß Sie unterschiedliche Typen von Socket-Verbindungen für RMI benutzen können (TCP/IP, UDP oder TCP mit einem anderen Protokoll wie beispielsweise SSL).

Wenn eine Client-Anwendung einen Remote Method Call durchführt, gelangt der Aufruf über Stub zu der Referenzebene, die die Argumente, falls erforderlich, zusammenstellt und gibt ihn dann über die Netzebene an den Server, wo die serverseitige Referenzebene die Argumente auseinandernimmt und sie an Skeleton und dann an die Server-Implementierung weiterleitet. Die Ausgabewerte für den Aufruf der Methode machen dann den Weg in umgekehrter Reihenfolge zurück zur Client-Seite.



Das Zusammenstellen und Weitergeben von Methodenargumenten ist einer der interessanteren Aspekte von RMI, da Objekte in etwas konvertiert werden müssen, das über das Netz weitergegeben werden kann. Diese Konvertierung nennt man *Serialisation*. So lange ein Objekt serialisiert werden kann, kann RMI es als einen Methodenparameter oder Ausgabewert benutzen. Serialisierbare Objekte beinhalten alle Java- Primitiv-Typen, Java-Remote-Objekte und jedes beliebige andere Objekt, das die *Serializable* -Schnittstelle (was viele der Klassen des Standard-1.2-JDK, wie beispielsweise alle AWT-Komponenten, beinhaltet) implementiert.

Als Methodenparameter oder Rückgabewerte benutzte Java-Remote-Objekte werden so wie lokale Objekte nach Referenz übergeben. Andere Objekte werden allerdings kopiert. Beachten Sie, daß dieses Verhalten die Art und Weise, in der Sie Ihre Java-Programme schreiben, beeinflußt, wenn diese Programme Remote Method Calls benutzen - Sie können beispielsweise ein Array nicht als ein Argument an eine Remote-Methode weitergeben, das Array von dem Remote-Objekt ändern lassen und erwarten, daß die lokale Kopie modifiziert wird. Dies stellt einen Unterschied zur Verhaltensweise von lokalen Kopien dar, wo alle Objekte als Referenzen weitergegeben werden.

RMI-Anwendungen erstellen

Um eine Anwendung zu erstellen, die RMI einsetzt, benutzen Sie die Klassen und Schnittstellen, die im Paket `java.rmi` definiert sind, das `java.rmi.server` für Server- seitige Klassen, `java.rmi.registry`, das die Klassen zur Lokalisierung und Registrierung der RMI-Server auf dem lokalen System enthält, und `java.rmi.dgc` für Garbage-Collection von verteilten Objekten beinhaltet. Das `java.rmi`-Paket selbst enthält die allgemeinen RMI-Schnittstellen, -Klassen und -Ausnahmen.

Um eine RMI-basierte Client-Server-Anwendung zu implementieren, definieren Sie zuerst eine Schnittstelle, die alle Methoden, die Ihr Remote-Objekt unterstützen wird, enthält. Die Methoden in dieser Schnittstelle müssen alle ein `throws RemoteException` -Statement beinhalten, das etwaige Netzprobleme behandelt, die zur Verhinderung der Kommunikation zwischen Client und Server führen können.

Der nächste Schritt besteht in der Implementierung der Remote-Schnittstelle in einer Server-seitigen Anwendung, die in der Regel die `UnicastRemoteObject`-Klasse erweitern. Innerhalb dieser Klasse können Sie die Methoden in der Remote-Schnittstelle implementieren und auch einen Security-Manager für diesen Server erstellen und installieren (um die Verbindungsherstellung von Random-Clients und die Durchführung nicht autorisierter Aufrufe von Methoden zu verhindern). Sie können natürlich den Security-Manager derart konfigurieren, daß er verschiedene Operationen genehmigt oder nicht genehmigt. In der Server-Anwendung »registrieren« Sie auch die Remote- Anwendung, die diese an einen Host und einen Port anbindet.

Auf der Client-Seite implementieren Sie eine einfache Anwendung, die die Remote- Schnittstelle benutzt und

Methoden in dieser Schnittstelle aufruft. Eine Klasse namens Naming (in java.rmi) ermöglicht dem Client eine transparente Verbindungsherstellung zum Server.

Wenn der Code geschrieben ist, können Sie ihn mit dem Standard-Compiler von Java kompilieren, allerdings gibt es noch einen weiteren Schritt auszuführen: benutzen Sie das Programm rmic zur Erzeugung der Ebenen Stubs und Skeletons, damit RMI effektiv zwischen den beiden Seiten des Prozesses arbeiten kann.

Das Programm rmiregistry wird schließlich dazu benutzt, die Server-Anwendung an das Netz selbst anzuschließen und es an einen Port anzubinden, so daß Remote-Verbindungen hergestellt werden können.

Natürlich stellt dies eine sehr vereinfachte Form des Prozesses zur Erstellung einer RMI- basierten Anwendung dar. Weitere Informationen über RMI und wie man die RMI-Klassen einsetzt, finden Sie in den Informationen über die Webseite von JavaSoft unter

<http://www.javasoft.com:80/products/JDK/1.2/docs/guide/rmi/index.html>

JDBC (Java Database Connectivity)

JDBC, das Java Database Connectivity API, definiert eine strukturierte Schnittstelle zu SQL-Datenbanken (Structured Query Language) - Industriestandard für relationale Datenbanken. Durch die Unterstützung von SQL ermöglicht JDBC Entwicklern die Interaktion und Unterstützung einer breiten Palette von Datenbanken. Dies bedeutet, daß die spezifischen Eigenschaften der zugrundeliegenden Datenbankplattform in bezug auf JDBC ziemlich irrelevant sind. Also gute Nachrichten für Java-Entwickler.

SQL-Datenbanken sind auf dem allgemein anerkannten SQL-Standard aufgebaute Datenbanken, der ein striktes Protokoll für den Zugriff auf und die Manipulation von Daten definiert.

Der Ansatz des JDBC-API für den Zugriff auf SQL-Datenbanken ist mit bestehenden Entwicklungstechniken für Datenbanken vergleichbar, so daß Interaktion mit einer SQL-Datenbank per JDBC nicht sehr unterschiedlich zu der Interaktion über traditionelle Datenbankwerkzeuge ist. Dies sollte Java-Programmierer, die schon einige Erfahrung mit Datenbanken haben, in bezug auf JDBC zuversichtlich stimmen. Das JDBC-API wurde bereits von vielen Industrieführern angenommen, inklusive einiger Lieferanten von Entwicklungswerkzeugen, die zukünftige Unterstützung von JDBC bei ihren Entwicklungsprodukten angekündigt haben.

Das JDBC-API beinhaltet Klassen für allgemeine Formen von SQL-Datenbanken wie Datenbankverbindungen, SQL-Anweisungen und Resultsets. Java-Programme mit JDBC sind in der Lage, das vertraute SQL-Programmierungsmodell der Ausgabe von SQL-Anweisungen und Verarbeitung der Ergebnisdaten zu benutzen. Das JDBC-API hängt weitgehend von einem Treibermanager ab, der mehrere Treiber, die die Verbindung zu verschiedenen Datenbanken herstellen, unterstützt. JDBC-Datenbanktreiber können entweder vollständig in Java geschrieben werden oder durch Verwendung systemeigener Methoden implementiert werden, um eine Brücke zwischen Java-Anwendungen und bestehenden Datenbankzugriffsbibliotheken zu schlagen.

JDBC beinhaltet ebenso eine Brücke zwischen JDBC und ODBC, der allgemeinen Schnittstelle zum Zugriff auf SQL-Datenbanken von Microsoft. Die JDBC-ODBC- Bridge ermöglicht es, JDBC-Treiber als ODBC-Treiber zu benutzen.

Die JDBC-Klassen sind als Bestandteil von Java 1.2 im java.sql-Paket enthalten und beinhalten Klassen zur Handhabung der Treiber, zum Verbindungsaufbau zwischen Datenbanken, zur Erstellung von SQL-Abfragen und Handhabung der Ergebnisse.

Die Webseite von JavaSoft enthält umfangreiche Informationen und Spezifikationen über JDBC unter <http://www.javasoft.com/products/JDK/1.2/docs/guide/jdbc/index.html>

Zusammenfassung

Als die Version 1 der Sprache Java 1995 veröffentlicht wurde, war die Sprache besser für die Applet-Programmierung geeignet als für den Entwurf großer Applikationen.

Mit Java 1.2 ist dies nicht mehr der Fall. Die Sprache bietet inzwischen eine robuste Unterstützung von Features, wie z.B. Entwurf von Softwarekomponenten, Remote Method Invocation, Datenbankbindung und Objektserialisation.

Das Material, das heute behandelt wurde, soll als Basis für die weitere Erforschung der Pakete und Klassen, die diese Features anbieten, dienen. Sobald Sie die Grundlagen der Sprache Java gemeistert haben, sind Sie bereit, die fortgeschritteneren Themen in Form von Büchern, Kursen oder Ihrer eigenen Entdeckungsreise durch das Java-API in Angriff zunehmen.

In den nächsten zwei Tagen werden wir Ihre Einführung in die Sprache Java abrunden. Sie werden lernen, wie Sie grafische Benutzeroberflächen mit Swing, der neuen Lösung für den Entwurf grafischer Benutzeroberflächen von Java 1.2, erstellen und wie Sie diese Oberflächen in funktionierende Applikationen verwandeln.

Fragen und Antworten

Frage:

showStatus() funktioniert nicht in meinem Browser. Wie kann ich meinen Lesern Statusinformationen ausgeben?

Antwort:

Wie Sie in dem Abschnitt über showStatus() gelernt haben, liegt es absolut am Browser, ob er showStatus() unterstützt. Wenn Sie in Ihrem Applet ein statusähnliches Verhalten benötigen, können Sie eine Statusbeschriftung (Label) im Applet erstellen, die auf die auszugebenden Informationen aktualisiert wird.

Frage:

Ich habe versucht, zwischen zwei Applets in meiner Webseite unter Verwendung der Methoden getAppletContext() und getApplet() zu kommunizieren. Meine Applets stürzen mit NullPointerException Fehlermeldungen ab. Was bedeutet das?

Antwort:

Der von mir beschriebene Mechanismus bezüglich der Kommunikation zwischen Applets beruht darauf, wie es laut SUN und der Java-Klassenbibliothek funktionieren sollte. Es verhält sich wie bei showStatus(): es hängt von dem jeweiligen Browser ab, ob er diesen Mechanismus unterstützt oder nicht bzw. ob er ihn korrekt ausführt. Die vorherige Version 3.0 von Netscape und Internet Explorer haben beide eigenartige Probleme mit Inter-Applet-Kommunikation.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 3

Tag 20

Benutzerschnittstellen mit Swing

In letzten zwei Tagen dieses Buches werden Sie mit einer Reihe von Klassen arbeiten, die sich Swing nennen. Mit diesen können Sie Benutzerschnittstellen mit einem Stil, der als Metal bezeichnet wird, implementieren.

Das klingt fast so, als ob einer bei JavaSoft entweder ein Musikfan oder ein frustrierter Musiker ist. Die nächste Technologie, die sich den Namen einer Musikrichtung entleiht, sollte Ska, Hiphop oder Bierfaß-Polka heißen.

Swing bietet gegenüber seinem Vorgänger eine wesentlich verbesserte Funktionalität - neue Komponenten, erweiterte Features der Komponenten, bessere Ereignisbehandlung und ein auswählbares Look and Feel.

Heute verwenden Sie Swing, um Java-Benutzerschnittstellen zu erstellen, und morgen lernen Sie, wie Sie diese Schnittstellen in vollwertige Programme verwandeln.



Swing ist auch als separates Add-On für Java 1.1 verfügbar. Wenn Sie ein Applet oder eine Applikation mit dem JDK 1.1 anstelle des JDK 1.2 schreiben, dann können Sie die Swing-Klassen verwenden, indem Sie sich die 1.1-kompatible Version der Java Foundation Classes von JavaSoft herunterladen. Momentan stehen diese auf der folgenden Seite zur Verfügung:

<http://java.sun.com/products/jfc/>

Da Swing in Java 1.2 vollständig integriert wurde, müssen Sie Swing nicht extra herunterladen, wenn Sie diese Version des JDK verwenden.

Die Vorteile von Swing

Look and Feel ist ein Ausdruck, der oft bei Erklärungen zur Programmierung von Benutzerschnittstellen verwendet wird. Die Bedeutung ist weitestgehend selbsterklärend - der Begriff beschreibt, wie eine Benutzerschnittstelle aussieht (Look) und wie sie sich für den Benutzer »anfühlt« (Feel) - anfühlen natürlich in bezug auf die Bedienung der Schnittstelle.

Swing ermöglicht es einem Java-Programm, ein unterschiedliches Look and Feel zu verwenden, gesteuert vom Programm selbst oder sogar durch den Benutzer.

Dieses Feature stellt die dramatischste Veränderung gegenüber dem AWT dar. Swing ermöglicht es Ihnen, ein Java-Programm mit einer Benutzerschnittstelle im Stil des jeweiligen Betriebssystems - z.B. Windows oder Solaris - oder einem Stil, der einzigartig bei Java ist (Dubbed Metal), zu erstellen.

Swing-Komponenten sind, anders als ihre Vorläufer in den vorherigen Versionen von Java, komplett in Java implementiert. Das hat eine bessere Kompatibilität zwischen den Programmen über unterschiedliche

Plattformen hinweg zur Folge, als dies bei Programmen, die Sie mit dem Abstract Windowing Toolkit erzeugt haben, der Fall ist.

Alle Elemente von Swing sind Bestandteil des Paketes `javax.swing`. Um eine Swing- Klasse zu verwenden, müssen Sie entweder eine `import`-Anweisung für die jeweilige Klasse oder eines für alle Klassen des Paketes wie im folgenden verwenden:

```
import javax.swing.*;
```

Der Prozeß bei der Verwendung der Swing-Komponenten ist nicht anders als bei Komponenten des Abstract Windowing Toolkit. Sie erzeugen eine Komponente, indem Sie deren Konstruktor aufrufen, Methoden der Komponente aufrufen, falls diese für die korrekte Einrichtung nötig sind, und die Komponente in einen Container einfügen.



Swing verwendet dieselbe Infrastruktur bei den Klassen, wie das beim AWT der Fall ist, wodurch es möglich wird, Swing- und AWT-Komponenten auf derselben Benutzerschnittstelle zu mischen. Allerdings werden die beiden Komponententypen zusammen in einem Container in manchen Fällen nicht korrekt dargestellt. Um diese Probleme zu vermeiden, ist es am besten, ausschließlich ein System in einem Programm zu verwenden.

Swing-Komponenten sind alle Subklassen der Klasse `JComponent`.

Ein Applikationsgerüst

Der erste Schritt bei der Erstellung einer einfachen Swing-Applikation ist es, eine Klasse zu erzeugen, die eine Subklasse der Klasse `JFrame` ist. Die Klasse `JFrame` ist eine Erweiterung der Klasse `Frame` und kann auf ähnliche Art und Weise verwendet werden.

Der Code in Listing 20.1 kann als Gerüst für jede beliebige Applikation dienen, die Sie erstellen und die ein Hauptfenster verwendet.

Listing 20.1: Der gesamte Quelltext von `Framework.java`

```
1: import java.awt.GridLayout;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
5: public class Framework extends JFrame {
6:
7:     public Framework() {
8:         super("Application Title");
9:
10:        // Hier die Komponenten einfügen
11:    }
12:
13:    public static void main(String[] args) {
14:        JFrame frame = new Framework();
15:
```

```

16:         WindowListener l = new WindowAdapter() {
17:             public void windowClosing(WindowEvent e) {
18:                 System.exit(0);
19:             }
20:         };
21:         frame.addWindowListener(l);
22:
23:         frame.pack();
24:         frame.setVisible(true);
25:     }
26: }

```

Diese Applikation ist eine Subklasse von JFrame. Die gesamte Arbeit zur Erstellung der Benutzerschnittstelle des Frames wird im Konstruktor Framework() erledigt.

Zeile 8 des Konstruktors gibt einen Text für die Titelleiste des Frame an. Dazu wird die Methode super(String) verwendet. Die Benutzerschnittstelle sollte in diesem Konstruktor erstellt werden - hier können Komponenten in Container eingefügt werden und Container in den Frame.

In der main()-Methode der Applikation passiert folgendes:

- Zeile 14 - Der Konstruktor Framework() wird dazu verwendet, eine neue Instanz der Klasse JFrame zu erzeugen. Diese Instanz stellt das Hauptfenster der Applikation dar.
- Zeilen 16 bis 21 - Diese Standard-Ereignisbehandlung kümmert sich darum, die Applikation zu schließen, wenn der Frame geschlossen wird. Über Ereignisbehandlungsroutinen, wie WindowListener lernen Sie morgen mehr.
- Zeile 23 - Hier wird die Methode pack() des Frame aufgerufen. Diese Methode sorgt dafür, daß die Größe des Frames so weit reduziert wird, daß alle Komponenten darin gerade noch Platz finden. Durch die Verwendung von pack() können Sie Komponenten in den Frame einfügen und sich sicher sein, daß für diese Platz zur Verfügung steht.
- Zeile 24 - Macht den Frame mit der Methode setVisible(boolean) sichtbar. Wenn das Argument false anstelle von true ist, dann wird der Frame unsichtbar.

Obwohl Sie dieses Gerüst erfolgreich kompilieren können, erzeugt es nichts Nützliches - das Fenster öffnet sich mit der minimal möglichen Größe, und Sie werden noch nicht einmal die Titelleiste in voller Länge sehen können.

Komponenten müssen hinzugefügt werden, bevor das Ganze beginnt, wie eine richtige Applikation auszusehen.

Komponenten in einen Swing-Frame einfügen



Die Arbeit mit einem JFrame-Objekt ist komplizierter als mit seinem AWT-Gegenstück. Anstatt Container und Komponenten direkt in den Frame einzufügen, müssen Sie diese in einen Zwischen-Container, der als *content pane* (*Inhaltsbereich*) bezeichnet wird, einfügen.

Ein JFrame ist in verschiedene Bereiche (panes) unterteilt. Der Hauptbereich, mit dem Sie arbeiten, ist der Inhaltsbereich. Dieser repräsentiert die gesamte Fläche des Frame, in die Komponenten eingefügt werden

können.

Um eine Komponente in den Inhaltsbereich einzufügen, tun Sie folgendes:

- Erzeugen Sie ein JPanel-Objekt (die Swing-Version eines Panels).
- Fügen Sie alle Komponenten (können auch Container sein) in das JPanel-Objekt über dessen add(Component)-Methode ein.
- Machen Sie dieses JPanel-Objekt über die Methode setContentPane(Container) zum Inhaltsbereich. Das JPanel-Objekt sollte das einzige Argument sein.

Das Programm in Listing 20.2 verwendet das Applikationsgerüst und fügt eine Schaltfläche in den Inhaltsbereich des Frame ein. Die Schaltfläche wird mit der Klasse JButton erzeugt, die Swing-Version einer anklickbaren Schaltfläche.

Listing 20.2: Der gesamte Quelltext von Swinger.java

```

1: import java.awt.GridLayout;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
5: public class Swinger extends JFrame {
6:
7:     public Swinger() {
8:         super("Swinger");
9:
10:         String note = "I receive a disproportionate amount of " +
11:             "joy from being clicked. Please interact with me.";
12:         JButton hotButton = new JButton(note);
13:
14:         JPanel pane = new JPanel();
15:         pane.add(hotButton);
16:
17:         setContentPane(pane);
18:     }
19:
20:     public static void main(String[] args) {
21:         JFrame frame = new Swinger();
22:
23:         WindowListener l = new WindowAdapter() {
24:             public void windowClosing(WindowEvent e) {
25:                 System.exit(0);
26:             }
27:         };
28:         frame.addWindowListener(l);
29:
30:         frame.pack();
31:         frame.setVisible(true);
32:     }
33: }
```

Abbildung 20.1 zeigt die Ausgabe dieser Applikation.

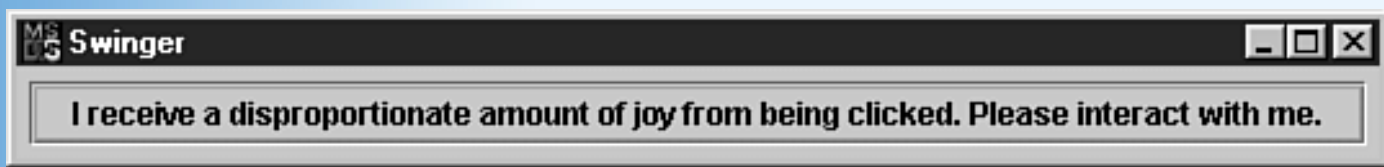


Abbildung 20.1: Die Swinger-Applikation

Das einzige neue Material in Listing 20.2 findet sich in den Zeilen 10 bis 17. Hier passiert folgendes:

- Zeilen 12 und 13 - Ein JButton-Objekt wird mit einem String als Label erzeugt. Dieses Vorgehen entspricht dem beim Konstruktor der Button-Klasse des AWT.
- Zeilen 14 und 15 - Ein JPanel-Objekt wird erstellt, und die Schaltfläche wird diesem Panel hinzugefügt.
- Zeile 17 - Mit der Methode setContentPanel(Container) wird das Panel zum Inhaltsbereich des Frame gemacht.

Sobald Sie den Inhaltsbereich für einen Frame festgelegt haben, verwenden Sie Methoden wie `setLayout(LayoutManager)` und `add(Component)` für diesen Bereich. Sie können diese Methoden nicht direkt für den Frame selbst aufrufen.



Dies trifft ebenfalls für Applets zu, die in Swing mit der Klasse `JApplet` implementiert sind. Sie müssen ein `JPanel`-Objekt erzeugen, Komponenten in dieses einfügen und dann das Panel zum Inhaltsbereich des Applet machen. Beachten Sie aber bitte, daß alle Applets, die Sie mit Swing und Java 1.2 erstellen, nicht mit Webbrowsern zusammenarbeiten, die lediglich Java 1.02 unterstützen. Sie müssen außerdem einen Weg finden, die Swing-Klassen für das Applet verfügbar zu machen. Die Klassen benötigen zusammen mit den Dateien, die zu dem Applet gehören, viel Zeit für den Download von der Website.

Der Umgang mit Swing

Zu jeder Komponente des Abstract Windowing Toolkit, die Sie bisher kennengelernt haben, gibt es entsprechende Swing-Komponenten. In den meisten Fällen gibt es einen Konstruktor für eine Swing-Komponente, der dem AWT-Gegenstück entspricht, so daß Sie nichts Neues lernen müssen, um mit Swing zu arbeiten.



Für viele Komponenten gibt es auch neue Konstruktoren, die ein `Icon`-Objekt als Argument erwarten. Ein *Icon* ist eine kleine Grafik, normalerweise im GIF-Format, die auf einer Schaltfläche, einem Label oder anderen Elementen der Benutzerschnittstelle verwendet werden kann, um diese Komponente zu identifizieren. In Form von Dateiordnern sehen Sie Icons auf grafischen Betriebssystemen wie Windows 95 und MacOS ständig.

Ein `Icon`-Objekt wird ganz ähnlich wie ein `Image`-Objekt erzeugt. Der Konstruktor erwartet den Dateinamen der Grafik oder die URL als einziges Argument. Das folgende Beispiel lädt ein Icon aus der Datei `unabom.gif` und erzeugt ein `JButton`-Objekt mit diesem Icon als Label. Abbildung 20.2 zeigt das

Ergebnis.

```

ImageIcon una = new ImageIcon("unabom.gif");
JButton button = new JButton(una);

JPanel pane = new JPanel();
pane.add(button);

setContentPane(pane);

```



Abbildung 20.2:
Ein Icon auf einem JButton-Objekt



Das Unabomber-Icon stammt von Jeffrey Zeldmans Icon-Sammlung »Pardon My Icons!«. Diese beinhaltet Hunderte von Icons, die Sie in Ihren eigenen Projekten verwenden können. Wenn Sie nach Icons suchen, mit denen Sie in Swing-Applikationen spielen können, dann sollten Sie sich mal auf der Pardon-My-Icons!-Website unter der folgenden Adresse umsehen:

<http://www.zeldman.com>

Der Quellcode zu diesem Beispiel steht auf der CD-ROM zu diesem Buch zur Verfügung. Der Name der Datei ist UnaButton.java, und das Icon befindet sich in der Datei unabom.gif.

Labels

Labels werden in Swing mit der Klasse JLabel implementiert. Die Funktionalität ist mit der der AWT-Labels vergleichbar. Allerdings können Sie inzwischen Icons verwenden. Zusätzlich kann die Ausrichtung eines Labels mit einer von 5 Klassenvariablen aus der Klasse SwingConstants festgelegt werden: LEFT, CENTER, RIGHT, LEADING oder TRAILING.

Unter anderem können Sie die folgenden Konstruktoren verwenden:

- JLabel(String, int) - Ein Label mit dem angegebenen Text und Ausrichtung.
- JLabel(String, Icon, int) - Ein Label mit dem angegebenen Text, Icon und Ausrichtung.

Schaltflächen

Wie Sie bereits gelernt haben, werden Swing-Schaltflächen über die Klasse JButton implementiert. Auf diesen kann sich ein Text-Label befinden, wie das bei AWT-Buttons der Fall ist, ein Icon-Label oder eine Kombination aus beiden.

Unter anderem können Sie die folgenden Konstruktoren verwenden:

- JButton(String) - Eine Schaltfläche mit dem angegebenen Text.
- JButton(Icon) - Eine Schaltfläche mit dem angegebenen Icon.

- JButton(String, Icon) - Eine Schaltfläche mit dem angegebenen Text und Icon.

Textfelder

Textfelder werden in Swing über die Klasse JTextField implementiert. Ein Unterschied zwischen diesen Textfeldern und ihren Gegenstücken aus dem AWT ist, daß die Methode setEchoChar(char) zur Verdeckung des eingegebenen Textes in JTextField nicht unterstützt wird.

Unter anderem können Sie die folgenden Konstruktoren verwenden:

- JTextField(int) - Ein Textfeld mit der angegebenen Breite.
- JTextField(String, int) - Ein Textfeld mit dem angegebenen Text und der angegebenen Breite.

Die Klasse JPasswordField wird verwendet, um ein Textfeld zu erzeugen, das den eingegebenen Text mit einem Zeichen verdeckt. Diese Klasse verfügt über dieselben Konstruktoren wie JTextField: JPasswordField(int) und JPasswordField(String, int).

Nachdem Sie ein Kennwortfeld erzeugt haben, können Sie die Methode setEchoChar(char) darauf anwenden, um das Zeichen festzulegen, mit dem die Eingabe verdeckt wird.

Mehrzeilige Textfelder (Textbereiche)

Textbereiche werden in Swing mit der Klasse JTextArea implementiert. Diese besitzt die folgenden Konstruktoren:

- JTextArea(int, int) - Ein Textbereich mit der angegebenen Anzahl von Zeilen und Spalten.
- JTextArea(int, int) - Ein Textbereich mit dem angegebenen Text und der angegebenen Anzahl von Zeilen und Spalten.

Kontrollkästchen und Optionsfelder

Die Klasse JCheckBox stellt die Implementierung von Kontrollkästchen in Swing dar. Die Funktionalität entspricht der des AWT-Gegenstücks mit der Erweiterung, daß Icons als Labels unterstützt werden.

Unter anderem können Sie die folgenden Konstruktoren verwenden:

- JCheckBox(String) - Ein Kontrollkästchen mit dem angegebenen Text-Label.
- JCheckBox(String, boolean) - Ein Kontrollkästchen mit dem angegebenen Text-Label. Das Kontrollkästchen ist markiert, wenn das zweite Argument true ist.
- JCheckBox(Icon) - Ein Kontrollkästchen mit dem angegebenen Icon-Label.
- JCheckBox(Icon, boolean) - Ein Kontrollkästchen mit dem angegebenen Icon-Label. Das Kontrollkästchen ist markiert, wenn das zweite Argument true ist.
- JCheckBox(String, Icon, boolean) - Ein Kontrollkästchen mit dem angegebenen Text-Label und Icon-Label. Das Kontrollkästchen ist markiert, wenn das zweite Argument true ist.

Kontrollkästchengruppen werden in Swing mit der Klasse ButtonGroup implementiert. Wie Sie gesehen haben, kann in einer Kontrollkästchengruppe immer nur ein Kontrollkästchen zugleich markiert sein. Sie erzeugen ein ButtonGroup-Objekt und fügen Kontrollkästchen mit der add(Component)-Methode in die Gruppe ein.

Optionsfelder werden in Swing über die Klasse JRadioButton implementiert. Die Konstruktoren entsprechen denen der Klasse JCheckBox.

Die Namensänderung von CheckboxGroup zu ButtonGroup spiegelt die erweiterte Funktionalität wider -

Schaltflächen und Optionsfelder können ebenfalls hiermit gruppiert werden.

Listenfelder

Listenfelder, wie sie im AWT mit der Klasse `Choice` erzeugt wurden, sind nur eine Möglichkeit, die Sie mit der Klasse `JComboBox` von Swing haben.

Ein Listenfeld wird über die folgenden Schritte erzeugt:

1. Der Konstruktor `JComboBox()` wird ohne Argumente aufgerufen.
2. Mit der Methode `addItem(Object)` des erzeugten Objekts werden der Liste Einträge hinzugefügt.
3. Die Methode `setEditable(boolean)` des Objekts wird mit `false` als Argument aufgerufen.

Diese letzte Methode sorgt dafür, daß ein Listenfeld entsteht - die einzigen Wahlmöglichkeiten für den Benutzer sind die Einträge, die zuvor in die Liste eingefügt wurden. Falls das `JComboBox`-Objekt editierbar ist, kann der Benutzer Text als Alternative zur Auswahl eines Eintrages in der Liste eingeben. Diese Kombination gibt den Kombinationslistenfeldern ihren Namen.

Bildlaufleisten

Bildlaufleisten werden in Swing über die Klasse `JScrollBar` implementiert. Die Funktionalität ist identisch mit der der AWT-Bildlaufleisten. Die folgenden Konstruktoren können Sie verwenden:

- `JScrollBar(int)` - Eine Bildlaufleiste mit der angegebenen Orientierung.
- `JScrollBar(int, int, int, int)` - Eine Bildlaufleiste mit der angegebenen Orientierung, dem angegebenen Startwert, Schiebergröße, Minimal- und Maximalwert.

Die Orientierung wird über die Variablen `HORIZONTAL` bzw. `VERTICAL` der Klasse `SwingConstants` festgelegt.

Ein Beispiel: die Applikation `SwingColorTest`

Eines der Projekte an Tag 14 war das Applet `ColorTest`, das die Auswahl einer Farbe über deren sRGB- oder HSB-Werte ermöglichte.

Das nächste Projekt erzeugt eine grafische Oberfläche für dieses Projekt mit Swing. Anstatt eines Applets erstellen wir hier eine Applikation. Die Methoden zur Ereignisbehandlung werden Sie morgen erzeugen.

Listing 20.3: Der gesamte Quelltext von `SwingColorTest.java`

```

1: import java.awt.*;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
5: public class SwingColorTest extends JFrame {
6:     SwingColorControls RGBcontrols, HSBcontrols;
7:     JPanel swatch;
8:
9:     public SwingColorTest() {
10:         super("Color Test");
11:
12:         JPanel pane = new JPanel();
13:         pane.setLayout(new GridLayout(1, 3, 5, 15));

```

```

14:         swatch = new JPanel();
15:         swatch.setBackground(Color.black);
16:         RGBcontrols = new SwingColorControls(this, "Red",
17:             "Green", "Blue");
18:         HSBcontrols = new SwingColorControls(this, "Hue",
19:             "Saturation", "Brightness");
20:         pane.add(swatch);
21:         pane.add(RGBcontrols);
22:         pane.add(HSBcontrols);
23:
24:         setContentPane(pane);
25:     }
26:
27:     public static void main(String[] args) {
28:         JFrame frame = new SwingColorTest();
29:
30:         WindowListener l = new WindowAdapter() {
31:             public void windowClosing(WindowEvent e) {
32:                 System.exit(0);
33:             }
34:         };
35:         frame.addWindowListener(l);
36:
37:         frame.pack();
38:         frame.setVisible(true);
39:     }
40:
41:     public Insets getInsets() {
42:         return new Insets(10, 10, 10, 10);
43:     }
44: }
45:
46: class SwingColorControls extends JPanel {
47:     SwingColorTest frame;
48:     JTextField tfield1, tfield2, tfield3;
49:
50:     SwingColorControls(SwingColorTest parent,
51:         String l1, String l2, String l3) {
52:
53:         frame = parent;
54:         setLayout(new GridLayout(3, 2, 10, 10));
55:         tfield1 = new JTextField("0");
56:         tfield2 = new JTextField("0");
57:         tfield3 = new JTextField("0");
58:         add(new JLabel(l1, JLabel.RIGHT));
59:         add(tfield1);
60:         add(new JLabel(l2, JLabel.RIGHT));
61:         add(tfield2);
62:         add(new JLabel(l3, JLabel.RIGHT));
63:         add(tfield3);

```



```

64:    }
65:
66:    public Insets getInsets() {
67:        return new Insets(10, 10, 0, 0);
68:    }
69: }

```

Abbildung 20.3 zeigt die Benutzerschnittstelle, die wir für diese Applikation entworfen haben. Obwohl die Schaltflächen und andere Komponenten ein anderes Look and Feel als das ColorTest-Applet an den Tag legen (darüber lernen Sie im Abschnitt »Das Look and Feel festlegen« mehr), funktioniert der Rest der Benutzerschnittstelle, wie das bei den Nicht-Swing-Gegenständen der Fall ist.

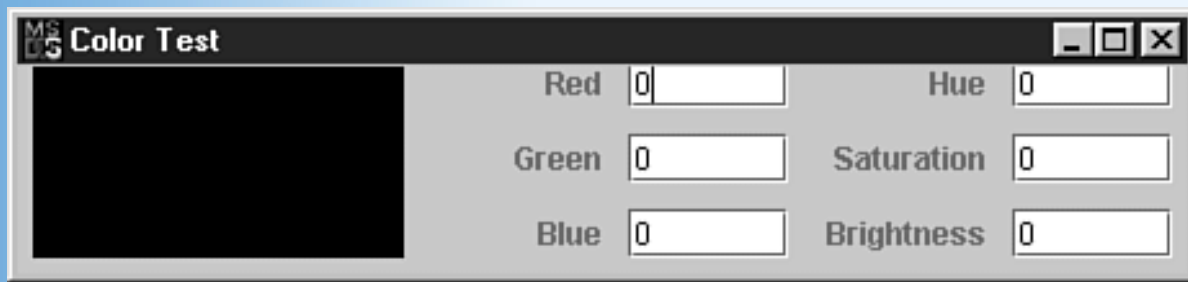


Abbildung 20.3:
Die SwingColorTest-Applikation

Das SwingColorTest-Programm verwendet das Applikationsgerüst, das weiter oben eingeführt wurde, so daß viele Teile des Programms bereits früher vorgestellt wurden.

Dieses Programm besteht aus drei Klassen: Der Hauptklasse SwingColorTest, der als private deklarierten Hilfsklasse SwingColorControls und einer Internen Klasse, die in den Zeilen 30 bis 34 definiert ist.

Sowohl die Klasse SwingColorTest als auch die Klasse SwingColorControls überschreiben die Methode getInsets(). Dies ermöglicht es den Komponenten, von dem Rand des Containers eine bestimmte Anzahl von Pixeln abgesetzt zu werden. Wie auch bei vielen anderen Aspekten von Swing wird dies in derselben Art und Weise unterstützt, wie das bei AWT-Komponenten der Fall ist.

Die Klasse SwingColorControls ist eine Subklasse der Klasse JPanel. Diese Klasse wurde für Swing aktualisiert. Dazu wurden die AWT-Komponenten Textfelder und Labels gegen Swing-Komponenten ausgetauscht. Ansonsten waren keine anderen Änderungen notwendig.

In der Klasse SwingColorTest aktualisierten die folgenden Änderungen den Code für die Benutzerschnittstelle, so daß anstelle des AWT Swing verwendet wird:

- Das Programm ist ein Subklasse von JFrame.
- Das swatch-Objekt, das die aktuell gewählte Farbe anzeigt, wird in Swing zu einem JPanel-Objekt anstelle eines Canvas-Objekts. In Swing gibt es kein Canvas-Objekt, statt dessen sollten Panels verwendet werden.
- Es wird ein JPanel-Objekt erzeugt, das Hauptinhaltsbereich des Frame sein soll.
- Die Komponenten swatch, RGBcontrols und HSBcontrols werden anstatt dem Hauptfenster des Programms dem Inhaltsbereich hinzugefügt.

In vielen Fällen kann eine Benutzerschnittstelle, die mit dem Abstract Windowing Toolkit erzeugt wurde, mit wenigen wesentlichen Änderungen mit Swing implementiert werden. Wenn Sie ein Applet, das mit Java 1.02 geschrieben wurde, in ein Swing-Programm für Java 1.2 konvertieren wollen, dann kommen größere Änderungen im Bereich der Ereignisbehandlungsmethoden, die morgen behandelt werden, auf

Sie zu.

Neue Features von Swing

Zusätzlich zu Komponenten und Containern, die die Funktionalität des Abstract Windowing Toolkit erweitern, bietet Swing noch viele Features, die vollkommen neu sind. Dazu gehören das definierbare Look and Feel, Tastatur-Mnemonics (Tastaturkürzel), ToolTips und Standarddialogfelder.

Das Look and Feel festlegen

Sie sind bereits mit dem Konzept der Layout-Manager vertraut - Klassen, die die Anordnung von Komponenten auf einer Benutzerschnittstelle kontrollieren.

Swing besitzt einen Manager für Benutzerschnittstellen, der das Look and Feel von Komponenten kontrolliert - die Art, in der Schaltflächen, Labels und andere Elemente auf dem Bildschirm dargestellt werden.

Das Management des Look and Feel wird von der Klasse UIManager im Paket übernommen. Die Auswahlmöglichkeiten, die Ihnen zur Verfügung stehen, hängen davon ab, welche Java-Entwicklungsumgebung Sie verwenden. Folgendes steht zur Verfügung:

- Das Windows-95- und Windows-NT-Look-and-Feel.
- Das Motif-X-Window-System-Look-and-Feel.
- Metal, das neue plattformübergreifende Look and Feel von Swing.

Die Klasse UIManager verfügt über die Methode `setLookAndFeel(LookAndFeel)`. Diese wird dazu verwendet, um das Look and Feel eines Programmes zu wählen.

Um ein `LookAndFeel`-Objekt zu erhalten, das Sie in der Methode `setLookAndFeel(LookAndFeel)` verwenden können, verwenden Sie die folgenden Methoden der Klasse UIManager:

- `getCrossPlatformLookAndFeelClassName()` - Diese Methode gibt ein `LookAndFeel`-Objekt zurück, das Javas plattformübergreifendes Metal-Look-and-Feel repräsentiert.
- `getSystemLookAndFeelClassName()` - Diese Methode gibt ein `LookAndFeel`-Objekt zurück, das das Look and Feel Ihres Systems repräsentiert.

Die Methode `setLookAndFeel(LookAndFeel)` wirft eine Ausnahme aus, wenn sie das Look and Feel nicht setzen kann.

Die folgenden Anweisungen können in einem beliebigen Programm dazu verwendet werden, Metal als Look and Feel festzulegen:

```
try {
    UIManager.setLookAndFeel(
        UIManager.getCrossPlatformLookAndFeelClassName());
} catch (Exception e) {
    System.err.println("Can't set look and feel: " + e);
}
```

Um das Look and Feel Ihres Systems zu wählen, verwenden Sie die Methode `getSystemLookAndFeelClassName()` als Argument für die Methode `setLookAndFeel()` im vorigen Beispiel. Dies erzeugt unterschiedliche Ergebnisse auf unterschiedlichen Betriebssystemen. Ein Windows-95-Benutzer erhält über `getSystemLookAndFeelClassName()` das Look and Feel von Windows 95. Ein Unix-Anwender würde das Motif Look and Feel erhalten.

Tastatur-Mnemonics



Ein Tastatur-Mnemonic (auch Tastaturkürzel oder Tastaturshortcut genannt) ist eine Tastenfolge, die zur Steuerung einer Komponente der Benutzerschnittstelle verwendet werden kann. Dies stellt einen Weg zur Verfügung, mit dem ein Programm ohne Maus verwendet werden kann. Tastatur, Mnemonics sind Teil der Unterstützung von Accessibility-Features durch Swing - neue Klassen, die es Blinden oder anderen Leuten mit Behinderungen ermöglichen, ein Java-Programm auszuführen.

Tastatur-Mnemonics simulieren eine Mausektion, sobald sie verwendet werden. Die Art, wie Mnemonics verwendet werden, unterscheidet sich je nach Plattform. Auf einem Computer, auf dem Windows 95 installiert ist, wird ein Tastatur-Mnemonic verfügbar, indem man die ALT-Taste in Kombination mit einer anderen Taste drückt.

Tastatur-Mnemonics werden über den Aufruf der Methode `setMnemonic(char)` der Komponente, die über dieses Tastatur-Mnemonic gesteuert werden soll, gesetzt. Das `char`-Argument ist die Taste, die als Teil des Tastatur-Mnemonic verwendet werden soll. Das folgende Beispiel erzeugt ein `JButton`-Objekt und verknüpft den Buchstaben 'i' mit der Schaltfläche:

```
JButton infoButton = new JButton("Information");
infoButton.setMnemonic('i');
```

Das Drücken der Tastenkombination `[ALT]+[I]` entspricht einem Anklicken der Schaltfläche.

ToolTips

Eine andere Methode, ein Programm benutzerfreundlicher zu gestalten, stellt die Verknüpfung von ToolTips mit Komponenten auf einer Benutzerschnittstelle dar. Sie werden eventuell schon mit ToolTips vertraut sein - kleine Textfahnen, die in einigen Programmen angezeigt werden, wenn Ihre Maus einige Sekunden über einer Komponente verharret.

ToolTips werden dazu verwendet, den Zweck einer Komponente zu beschreiben. Wenn Sie die Bedienung eines Programmes erlernen, stellen ToolTips - sofern sie implementiert wurden - eine exzellente Lernhilfe dar.

Um einen Tooltip für eine Komponente zu setzen, rufen Sie die Methode `setToolTipText(String)` für die jeweilige Komponente auf. Der String sollte eine knappe Beschreibung der Aufgabe der Komponente sein.

Das folgende Beispiel erzeugt eine `JScrollBar`-Komponente und verknüpft einen Tooltip mit dieser:

```
JScrollBar speed = new JScrollBar();
speed.setToolTipText("Move to set animation speed");
```

Der Text für den Tooltip ist auf eine Zeile begrenzt. Aus diesem Grund können Sie die Escape-Sequenz für einen Zeilenumbruch (`>\n<`) nicht verwenden, um den Text in mehrere Zeilen aufzuteilen.

Komponentenbeschreibungen und -namen

Eine weitere Möglichkeit, eine Benutzerschnittstelle leichter zugänglich zu machen, ist, einen Beschreibungstext anzubieten. Dies wird über einen zweistufigen Prozeß umgesetzt:

1. Über einen Aufruf der Methode `getAccessibleContext()` der Komponente das `AccessibleContext`-Objekt, das der Komponente zugewiesen ist, ermitteln.
2. Aufrufen der Methode `setAccessibleDescription(String)` des `AccessibleContext`-Objektes. Das `String`-Argument sollte die Beschreibung der Komponente sein.

Das folgende Beispiel setzt die Beschreibung eines `JButton`-Objekts:

```
JButton quit = new JButton("Quit");
quit.getAccessibleContext().setAccessibleDescription(
    "When you click this button, the program terminates.");
```

Die Methode `setAccessibleName(String)` funktioniert auf die gleiche Weise wie die Methode `setAccessibleDescription(String)`. Diese Methode kann dazu verwendet werden, einer Komponente einen Namen zu geben, der ihren Zweck kurz beschreibt. »Quit-Schaltfläche« wäre ein geeigneter Name für das `quit`-Objekt im vorigen Beispiel. Das nächste Beispiel setzt den Namen für das Textfeld `nm` auf "Namens-Feld":

```
JTextField nm = new JTextField();
nm.getAccessibleContext().setAccessibleName("Namens-Feld");
```

Standarddialogfelder

Die Klasse `JOptionPane` enthält einige Methoden, die zur Erzeugung von Standarddialogfeldern verwendet werden können: kleine Fenster, die eine Frage stellen, einen Benutzer warnen oder eine kurze wichtige Meldung anzeigen. Abbildung 20.4 zeigt ein Beispiel eines Dialogfeldes im Metal-Look-and-Feel.

Zweifellos haben Sie schon derartige Dialogfelder gesehen - wenn sich Ihr System aufhängt, zeigt ein Dialogfeld diese unerfreuliche Nachricht an, wenn Sie Dateien löschen, fragt Sie ein Dialogfeld, ob Sie dies wirklich tun wollen. Diese Fenster sind ein effektiver Weg, mit einem Benutzer zu kommunizieren, ohne den Overhead, neue Klassen zu erstellen, die diese Fenster repräsentieren, Komponenten in diese einzufügen und den Code für die Ereignisbehandlung der Benutzereingaben schreiben zu müssen. Alle diese Dinge werden automatisch erledigt, wenn eines der Standarddialogfelder von `JOptionPane` verwendet wird.

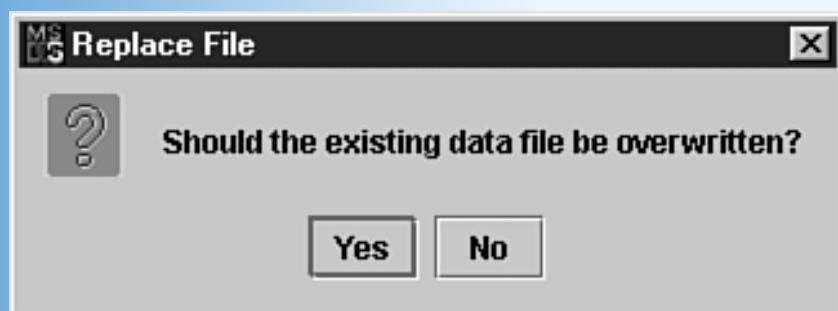


Abbildung 20.4: Ein Standarddialogfeld.

Es gibt vier Standarddialogfelder:

- `ConfirmDialog` - Ein Dialogfeld, das eine Frage stellt und die Schaltflächen Yes, No und Cancel

bietet, damit der Benutzer entsprechend antworten kann.

- **InputDialog** - Ein Dialogfeld, das zur Eingabe von Text auffordert.
- **MessageDialog** - Ein Dialogfeld, das eine Meldung anzeigt.
- **OptionDialog** - Ein Dialogfeld, das die drei anderen Dialogfelder kombiniert.

Jedes dieser Dialogfelder hat seine eigene Methode in der Klasse `JOptionPane`.

ConfirmDialog-Dialogfelder

Den einfachsten Weg, ein Yes/No/Cancel-Dialogfeld zu erzeugen, stellt der folgende Methodenaufruf dar: `showConfirmDialog(Component, Object)`). Das Argument `Component` gibt den Container an, dem das Dialogfeld zugeordnet sein soll. Diese Information wird dazu verwendet, festzulegen, wo am Bildschirm das Dialogfeld angezeigt werden soll. Wenn anstelle eines Containers `Null` verwendet wird oder wenn der Container kein `Frame`-Objekt ist, dann wird das Dialogfeld auf dem Bildschirm zentriert.

Das zweite Argument kann ein `String`, eine Komponente oder ein `Icon`-Objekt sein. Wenn es ein `String` ist, dann wird dieser Text in dem Dialogfeld angezeigt. Wenn es sich dagegen um eine Komponente oder ein `Icon` handelt, dann wird dieses Objekt anstelle einer Textmeldung angezeigt.

Diese Methode gibt einen von drei möglichen Integer-Werten zurück. Jeder davon ist eine Klassenvariable von `JOptionPane`: `YES_OPTION`, `NO_OPTION` und `CANCEL_OPTION`.

Das folgende Beispiel verwendet ein Dialogfeld vom Typ `ConfirmDialog` mit einer Textmeldung und speichert die Antwort in der Variablen `response`:

```
int response;
response = JOptionPane.showConfirmDialog(null,
    "Should I delete all of your irreplaceable personal files");
```

Eine andere Methode bietet mehr Optionen für Dialogfelder des Typs `ConfirmDialog`: `showConfirmDialog(Component, Object, String, int, int)`. Die ersten beiden Argumente entsprechen denen der anderen `showConfirmDialog()`-Methode. Die letzten drei werden im folgenden beschrieben:

- Ein `String`, der in der Titelleiste des Dialogfeldes angezeigt wird.
- Ein `Integer`, der angibt, welche Schaltflächen angezeigt werden. Dieser sollte einer der Klassenvariablen `YES_NO_CANCEL_OPTION` oder `YES_NO_OPTION` entsprechen.
- Ein `Integer`, der die Art des Dialogfeldes beschreibt. Dazu werden die Klassenvariablen `ERROR_MESSAGE`, `INFORMATION_MESSAGE`, `PLAIN_MESSAGE`, `QUESTION_MESSAGE` oder `WARNING_MESSAGE` verwendet. Dieses Argument wird dazu benutzt, das Icon, das in dem Dialogfeld neben dem Meldungstext angezeigt wird, festzulegen.

```
int response = JOptionPane.showConfirmDialog(null,
    "Error reading file. Want to try again?",
    "File Input Error",
    JOptionPane.YES_NO_OPTION,
    JOptionPane.ERROR_MESSAGE);
```

In Abbildung 20.5 sehen Sie das Dialogfeld, das das obige Beispiel erzeugt, im Windows-Look-and-Feel.

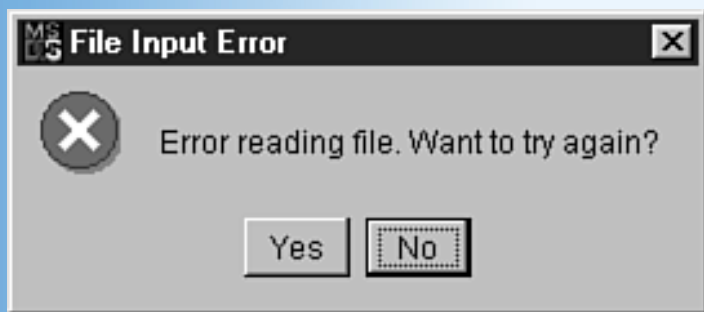


Abbildung 20.5:
Ein Dialogfeld vom Typ ConfirmDialog

InputDialog-Dialogfelder

Ein Dialogfeld des Typs InputDialog stellt eine Frage und verwendet ein Textfeld, um die Antwort zu speichern. Abbildung 20.6 zeigt ein Beispiel, das das Motif-Look-and-Feel verwendet.

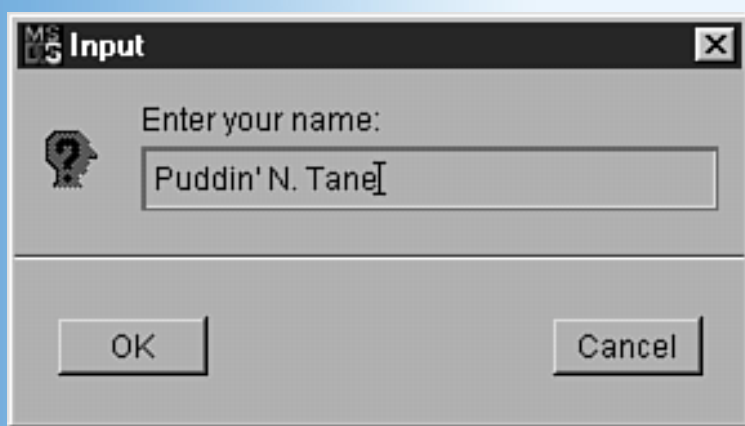


Abbildung 20.6:
Ein Dialogfeld vom Typ InputDialog

Der einfachste Weg, ein Dialogfeld des Typs InputDialog zu erstellen, ist ein Aufruf der Methode `showInputDialog(Component, Object)`. Die Argumente sind der Container, dem das Dialogfeld zugeordnet ist, und der String, die Komponente oder das Icon, das in dem Dialogfeld angezeigt werden soll.

Der Aufruf der Methode gibt einen String zurück, der die Antwort des Benutzers repräsentiert. Die folgende Anweisung erzeugt das Dialogfeld, das in Abbildung 20.6 gezeigt ist:

```
String response = JOptionPane.showInputDialog(null,
    "Enter your name:");
```

Sie können ein solches Dialogfeld auch mit der Methode `showInputDialog(Component, Object, String, int)` erzeugen. Die ersten beiden Argumente entsprechen dem kürzeren Methodenaufruf und die letzten beiden werden im folgenden erklärt:

- Der Titel, der in der Titelleiste des Dialogfeldes angezeigt werden soll.
- Eine von fünf Klassenvariablen, die die Art des Dialogfeldes beschreibt: `ERROR_MESSAGE`, `INFORMATION_MESSAGE`, `PLAIN_MESSAGE`, `QUESTION_MESSAGE` oder `WARNING_MESSAGE`.

Die folgende Anweisung erzeugt ein Dialogfeld vom Typ InputDialog mit der oben beschriebenen Methode:

```
String response = JOptionPane.showInputDialog(null,
```

```
"What is your ZIP code?",
"Enter ZIP Code",
JOptionPane.QUESTION_MESSAGE);
```

MessageDialog-Dialogfelder

Dialogfelder des Typs `MessageDialog` sind einfache Fenster, die Informationen anzeigen. In Abbildung 20.7 sehen Sie ein Beispiel im Metal-Look-and-Feel.

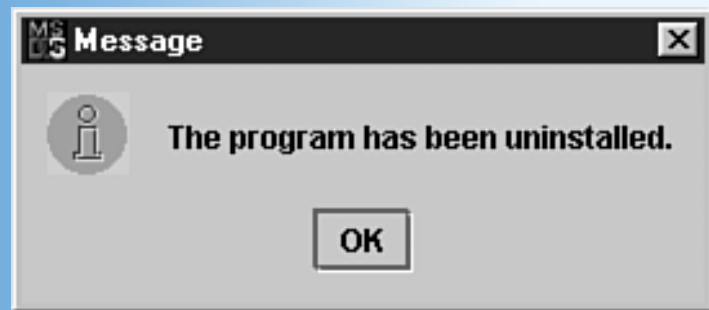


Abbildung 20.7:
Ein Dialogfeld vom Typ `MessageDialog`

Ein solches Dialogfeld kann über einen Aufruf der Methode `showMessageDialog(Component, Object)` erzeugt werden. Wie bei den anderen Dialogfeldtypen stellen die Argumente den zugeordneten Container und den anzuzeigenden String, die Komponente oder das Icon dar.

Anders als bei den anderen Dialogfeldtypen gibt diese Art Dialogfeld keinerlei Wert zurück. Die folgende Anweisung erzeugt das Dialogfeld aus Abbildung 20.7:

```
JOptionPane.showMessageDialog(null,
    "The program has been uninstalled.");
```

Zusätzlich können Sie ein Dialogfeld dieses Typs mit der Methode `showMessageDialog(Component, Object)` erzeugen. Die Anwendung entspricht der der Methode `showInputDialog()` mit denselben Argumenten. Einziger Unterschied: Die Methode gibt keinen Wert zurück.

Die folgende Anweisung erzeugt ein Dialogfeld des Typs mit dieser Methode:

```
String response = JOptionPane.showMessageDialog(null,
    "An asteroid has destroyed the Earth.",
    "Asteroid Destruction Alert",
    JOptionPane.WARNING_MESSAGE);
```

OptionDialog-Dialogfelder

Dialogfelder des Typs `OptionDialog` stellen die komplexeste Art dar. Diese Dialogfelder kombinieren die Features aller anderen Dialogfeldarten. Sie können mit der Methode `showOptionDialog(Component, Object, String, int, int, Icon, Object[], Object)` erzeugt werden.

Die Argumente der Methode haben die folgende Bedeutung:

- Der zugeordnete Container des Dialogfeldes.
- Der Text, das Icon oder die Komponente, die angezeigt werden soll.
- Ein String, der in der Titelleiste angezeigt werden soll.
- Die Art des Dialogfeldes. Dazu verwenden Sie die Klassenvariablen `YES_NO_OPTION`, `YES_NO_CANCEL_OPTION` oder das Literal `0`, falls andere Schaltflächen verwendet werden

sollen.

- Das Icon, das angezeigt werden soll. Dazu verwenden Sie die Klassenvariablen `ERROR_MESSAGE`, `INFORMATION_MESSAGE`, `PLAIN_MESSAGE`, `QUESTION_MESSAGE`, `WARNING_MESSAGE` oder das Literal 0, falls keines dieser Icons verwendet werden soll.
- Ein Icon-Objekt, das anstelle eines der Icons im vorherigen Argument angezeigt werden soll.
- Ein Array mit Objekten, das die Komponenten oder andere Objekte beinhaltet, die die Wahlmöglichkeiten repräsentieren, wenn weder `YES_NO_OPTION` noch `YES_NO_CANCEL_OPTION` verwendet wird.
- Das Objekt, das die Standardauswahl repräsentiert, wenn weder `YES_NO_OPTION` noch `YES_NO_CANCEL_OPTION` verwendet wird.

Die beiden letzten Argumente ermöglichen es Ihnen, in dem Dialogfeld eine große Anzahl von Wahlmöglichkeiten zu erstellen. Sie können ein Array mit Schaltflächen, Textfeldern oder sogar mit einer Mischung unterschiedlicher Komponenten als Objekt-Array erzeugen. Diese Komponenten werden mit dem `FlowLayout`-Layout-Manager angeordnet - es besteht keine Möglichkeit, einen anderen Layout-Manager für diesen Dialog zu wählen.

Das folgende Beispiel erzeugt ein Dialogfeld, das ein Array mit `JButton`-Objekten als Wahlmöglichkeit in dem Dialogfeld verwendet. Das `gender[2]`-Element wird als Standardauswahl festgelegt:

```
JButton[] gender = new JButton[3];
gender[0] = new JButton("Male");
gender[1] = new JButton("Female");
gender[2] = new JButton("None of Your Business");
int response = JOptionPane.showOptionDialog(null,
    "What is your gender?",
    "Gender",
    0,
    JOptionPane.INFORMATION_MESSAGE,
    null,
    gender,
    gender[2]);
```

Abbildung 20.8 zeigt das Dialogfeld, das sich aus diesem Code ergibt, im Motif-Look- and-Feel.

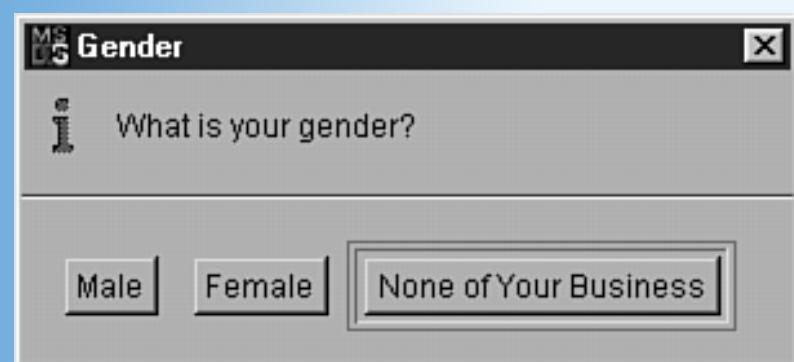


Abbildung 20.8:
Ein Dialogfeld vom Typ `OptionDialog`

Ein Beispiel: die Info-Applikation

Das nächste Projekt bietet eine Chance, eine Reihe von Dialogfeldern in einem funktionierenden Programm zu erleben. Die Info-Applikation verwendet Dialogfelder, um Informationen vom Benutzer zu ermitteln. Diese Informationen werden anschließend im Hauptfenster der Applikation in Textfeldern platziert.

Tippen Sie doch das Listing 20.4 ab, und kompilieren Sie das Ganze.

Listing 20.4: Der gesamte Quelltext von Info.java.

```

1: import java.awt.GridLayout;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
5: public class Info extends JFrame {
6:     private JLabel titleLabel = new JLabel("Title: ",
7:         SwingConstants.RIGHT);
8:     private JTextField title;
9:     private JLabel addressLabel = new JLabel("Address: ",
10:         SwingConstants.RIGHT);
11:     private JTextField address;
12:     private JLabel typeLabel = new JLabel("Type: ",
13:         SwingConstants.RIGHT);
14:     private JTextField type;
15:
16:     public Info() {
17:         super("Site Information");
18:
19:         // Site-Name
20:         String response1 = JOptionPane.showInputDialog(null,
21:             "Enter the site title:");
22:         title = new JTextField(response1, 20);
23:
24:         // Site-Adresse
25:         String response2 = JOptionPane.showInputDialog(null,
26:             "Enter the site address:");
27:         address = new JTextField(response2, 20);
28:
29:         // Site-Art
30:         String[] choices = { "Personal", "Commercial", "Unknown" };
31:         int response3 = JOptionPane.showOptionDialog(null,
32:             "What type of site is it?",
33:             "Site Type",
34:             0,
35:             JOptionPane.QUESTION_MESSAGE,
36:             null,
37:             choices,
38:             choices[0]);
39:         type = new JTextField(choices[response3], 20);
40:
41:         JPanel pane = new JPanel();

```

```

42:         pane.setLayout(new GridLayout(3, 2));
43:         pane.add(titleLabel);
44:         pane.add(title);
45:         pane.add(addressLabel);
46:         pane.add(address);
47:         pane.add(typeLabel);
48:         pane.add(type);
49:
50:         setContentPane(pane);
51:     }
52:
53:     public static void main(String[] args) {
54:         try {
55:             UIManager.setLookAndFeel(
56:                 UIManager.getSystemLookAndFeelClassName());
57:         } catch (Exception e) {
58:             System.err.println("Couldn't use the system "
59:                               + "look and feel: " + e);
60:         }
61:
62:         JFrame frame = new Info();
63:
64:         WindowListener l = new WindowAdapter() {
65:             public void windowClosing(WindowEvent e) {
66:                 System.exit(0);
67:             }
68:         };
69:         frame.addWindowListener(l);
70:
71:         frame.pack();
72:         frame.setVisible(true);
73:     }
74: }

```

Abbildung 20.9 zeigt das Hauptfenster der Applikation im systemeigenen Look and Feel (in diesem Screenshot ist es Windows). Drei Textfelder beinhalten Werte, die sie aus Dialogfeldern erhalten.

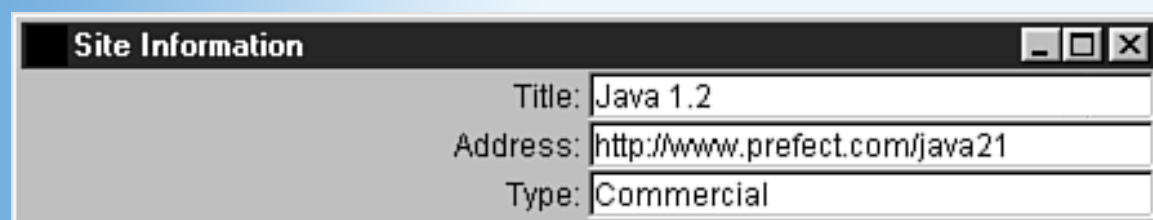


Abbildung 20.9:
Das Hauptfenster der Info-Applikation

Ein großer Teil dieser Applikation ist vorgefertigter Code, der bei jeder Swing-Applikation verwendet werden kann. Die folgenden Zeilen beziehen sich auf die Dialogfelder:

- Zeile 19 bis 22 - Ein Dialogfeld vom Typ `InputDialog` wird verwendet, um den Benutzer zur Eingabe des Namens der Site aufzufordern. Dieser Name wird im Konstruktor eines `JTextField`-Objektes verwendet. Dieser fügt den Text in das Textfeld ein.

- Zeile 24 bis 27 - Ein ähnliches Dialogfeld desselben Typs wird verwendet, um den Benutzer zur Eingabe der Adresse der Site aufzufordern. Dieser wird im Konstruktor eines weiteren JTextField-Objektes verwendet.
- Zeile 30 - Ein Array mit String-Objekten namens choices wird erzeugt und mit drei Werten initialisiert.
- Zeile 31 bis 38 - Ein Dialogfeld des Typs OptionDialog wird verwendet, um den Benutzer den Typ der Site auswählen zu lassen. Das choices-Array ist das siebte Argument, das drei Schaltflächen auf dem Dialogfeld mit den Strings in dem Array einrichtet: Personal, Commercial und Unknown. Das letzte Argument, choices[0], legt das erste Element des Arrays als Standardauswahl fest. Abbildung 20.10 zeigt dieses Dialogfeld.
- Zeile 39 - Die Antwort des Dialogfeldes - ein Integer, der das gewählte Array-Element identifiziert - wird in eine JTextField-Komponente mit dem Namen type gespeichert.

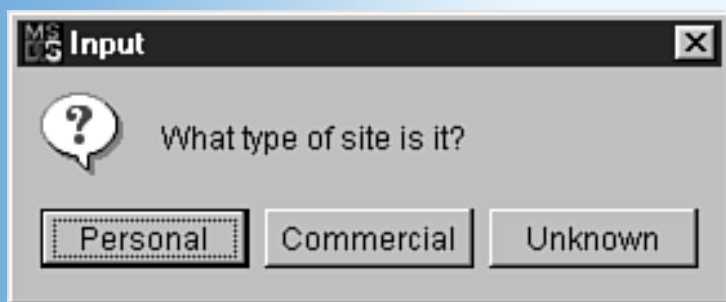


Abbildung 20.10:
Das Dialogfeld zum Abfragen des Typs der Site

Zusammenfassung

Nachdem viele hunderttausend Programmierer die Chance hatten, die ersten Versionen von Java zu verwenden, zielten die meisten Beschwerden auf das Abstract Windowing Toolkit ab. Obwohl es die Erstellung einer funktionellen Benutzerschnittstelle erlaubt, gab es einige Probleme, alle Benutzerschnittstellen auf unterschiedlichen Plattformen zum Laufen zu bringen. Außerdem wurden einige Elemente einer grafischen Benutzeroberfläche nicht vom AWT unterstützt.

Swing ist eine effektive Antwort auf diese Kritik. Es bietet ein ausgefeiltes System zur Erstellung grafischer Benutzeroberflächen, das sich für viele unterschiedliche Arten von Java-Programmen einsetzen läßt. Wenn Sie einen Blick in die Dokumentation von Swing werfen, werden Sie mehr als 30 verschiedene Komponenten finden.

Morgen werden Sie lernen, eine Benutzerschnittstelle in eine vollständige Applikation zu verwandeln.

Fragen und Antworten

Frage:
Kann eine Applikation auch ohne Swing erstellt werden?

Antwort:
Sicherlich - Swing ist lediglich eine Erweiterung des Abstract Windowing Toolkit, und Sie können auch unter Java 1.2 weiterhin das AWT für Applikationen verwenden. Sie sollten lediglich die Techniken zur Ereignisbehandlung, die Sie morgen lernen, anstelle der Techniken für Java-1.02-Applets verwenden. Ob Sie eine Applikation ohne Swing erstellen sollten, ist eine andere Frage. Es werden im AWT keine zu

Swing vergleichbaren Möglichkeiten angeboten. Mit Swing können Sie weitaus mehr Komponenten auf eine viel ausgefeiltere Art verwenden.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Woche 1**Tag 21****Ereignisbehandlung mit Swing**

In diesem Kapitel stehen zwei große Ereignisse an: Der Abschluß Ihrer dreiwöchigen Reise durch die Programmiersprache Java und die Ereignisse, die Sie in Swing-Programmen zu behandeln lernen.

Um eine funktionierende Java-Benutzeroberfläche in ein funktionierendes Java-Programm zu verwandeln, müssen Sie der Oberfläche beibringen, auf Ereignisse zu reagieren, die vom Benutzer ausgelöst werden.

Sie haben bereits mit Ereignissen zu tun gehabt, als Sie gelernt haben, Mausklicks und andere Benutzereingaben mit dem Abstract Windowing Toolkit zu behandeln. Dieses Wissen wurde verwendet, um Java-1.02-kompatible Applets zu erstellen.



Swing behandelt im Vergleich zu Java 1.02 Ereignisse anders, und zwar mit Klassen, die als *Event Listener* bezeichnet werden. Heute lernen Sie, wie Sie Listener aller Arten in Ihre Swing-Programme einfügen. Zu diesen Listnern gehören u.a. solche, die auf Aktionsereignisse, Mausereignisse und andere Interaktionen hören.

Wenn Sie durch sind, feiern Sie dieses Ereignis mit der Vervollständigung einer Java- Applikation mit den Swing-Klassen.

Das Hauptereignis

In dem Ereignisbehandlungssystem, das Sie letzte Woche kennengelernt haben, wurden Ereignisse mit einem Satz von Methoden, die allen Komponenten zur Verfügung stehen, behandelt. Methoden wie `mouseDown()`, `keyDown()` und `action()` können in jedem AWT-Programm, das diese Ereignisse behandeln will, überschrieben werden.

Dieses System zur Ereignisbehandlung bezieht sich nur auf Java 1.02, da in den Folgeversionen eine stark verbesserte Lösung für die Ereignisbehandlung angeboten wurde.

Das neue System verwenden Sie, um Swing-Applikationen zu erstellen.

Event Listener

Wenn eine Klasse im Ereignisbehandlungsmodell von Java 1.2 auf ein Ereignis reagieren will, muß diese eine Schnittstelle implementieren, die diese Ereignisse verarbeitet. Diese Schnittstellen werden als Event Listener bezeichnet.

Jeder Listener behandelt eine bestimmte Ereignisart. Eine Klasse kann so viele Listener implementieren, wie benötigt werden.

Die folgenden Event Listener stehen zur Verfügung:

- `ActionListener` - Aktionsereignisse, die durch einen Benutzer ausgelöst werden, der eine Aktion auf einer Komponente ausführt, wie z.B. einen Klick auf eine Schaltfläche.
- `AdjustmentListener` - Ereignisse, die erzeugt werden, wenn Werte mit einer Komponente eingestellt werden (wenn z.B. der Schieber einer Bildlaufleiste bewegt wird).

- **FocusListener** - Ereignisse, die erzeugt werden, wenn eine Komponente, wie z.B. ein Textfeld, den Eingabefokus erhält oder verliert.
- **ItemListener** - Ereignisse, die erzeugt werden, wenn ein Element, wie z.B. ein Kontrollkästchen, verändert wurde.
- **KeyListener** - Tastaturereignisse, die erzeugt werden, wenn ein Benutzer Text über die Tastatur eingibt.
- **MouseListener** - Mausereignisse, die erzeugt werden, wenn mit der Maus geklickt wird, die Maus den Bereich einer Komponente betritt bzw. diesen wieder verläßt.
- **MouseMotionListener** - Mausereignisse, über die die Bewegung der Maus über eine Komponente verfolgt wird.
- **WindowListener** - Ereignisse, die von Fenstern, wie dem Hauptfenster einer Applikation, erzeugt werden, wenn sie maximiert, minimiert, verschoben oder geschlossen werden.

Die folgende Klasse ist so deklariert, daß sie sowohl Aktions- als auch Textereignisse behandeln kann:

```
public class Suspense extends JFrame implements ActionListener,
    TextListener {
    // ...
}
```

Das Paket `java.awt.event` beinhaltet alle der elementaren Event Listener wie auch die Objekte, die spezielle Ereignisse repräsentieren. Um diese Klassen in Ihren Programmen zu verwenden, können Sie sie einzeln importieren oder eine Anweisung wie die folgende verwenden:

```
import java.awt.event.*;
```

Komponenten einrichten

Indem Sie eine Klasse zu einem Event Listener machen, haben Sie eine bestimmte Ereignisart festgelegt, auf die diese Klasse hört. Allerdings wird dies nie passieren, wenn Sie nicht einen zweiten Schritt folgen lassen: Es muß ein passender Listener in die Komponente eingefügt werden. Dieser Listener erzeugt die Ereignisse, wenn die Komponente verwendet wird.

Nachdem eine Komponente erzeugt wurde, können Sie eine der folgenden Methoden für die Komponente aufrufen, um dieser einen Listener zuzuordnen:

- **addActionListener()** - JButton-, JCheckBox-, JComboBox-, JTextField- und JRadioButton-Komponenten
- **addAdjustmentListener()** - JScrollBar-Komponenten
- **addFocusListener()** - Alle Swing-Komponenten
- **addItemListener()** - JButton-, JCheckBox-, JComboBox- und JRadioButton-Komponenten
- **addKeyListener()** - Alle Swing-Komponenten
- **addMouseListener()** - Alle Swing-Komponenten
- **addMouseMotionListener()** - Alle Swing-Komponenten
- **addWindowListener()** - Alle JWindow- und JFrame-Komponenten



Eine Komponente zu verändern, nachdem sie in einen Container eingefügt wurde, ist ein Fehler, der einem in einem Java-Programm sehr leicht unterlaufen kann. Sie müssen einer Komponente Listener hinzufügen und alle anderen Konfigurationen vornehmen, bevor Sie diese in irgendeinen Container einfügen. Andernfalls werden diese Einstellungen bei der Ausführung des Programms mißachtet.

Das folgende Beispiel erzeugt ein JButton-Objekt und verknüpft es mit einem Listener für Aktionsereignisse:

```
JButton zap = new JButton("Zap");
```

```
zap.addActionListener(this);
```

Alle der unterschiedlichen Methoden erwarten ein Argument: das Objekt, das auf Ereignisse dieser Art hört. Wenn Sie `this` verwenden, zeigt dies, daß die aktuelle Klasse der Event Listener ist. Sie könnten auch ein anderes Objekt angeben, solange dessen Klasse die geeigneten Listener-Schnittstellen implementiert.

Methoden für die Ereignisbehandlung

Wenn Sie eine Schnittstelle mit einer Klasse verknüpfen, muß diese Klasse auch alle Methoden der Schnittstelle implementieren.

Im Fall der Event Listener wird jede der Methoden automatisch aufgerufen, sobald das korrespondierende Benutzereignis eintritt.

Die ActionListener-Schnittstelle besitzt nur eine einzige Methode `actionPerformed()`. Alle Klassen, die ActionListener implementieren, müssen über diese Methode mit einer Struktur wie der folgenden verfügen:

```
public void actionPerformed(ActionEvent evt) {
    // Ereignis hier verarbeiten
}
```

Wenn nur eine Komponente in Ihrer grafischen Benutzeroberfläche einen Listener für Aktionsereignisse besitzt, kann diese `actionPerformed()`-Methode zur Behandlung der Ereignisse, die von dieser Komponente erzeugt werden, verwendet werden.

Wenn mehr als eine Komponente einen Listener für Aktionsereignisse hat, müssen Sie in der Methode herausfinden, welche Komponente verwendet wurde, und anschließend entsprechend darauf reagieren.

Sie haben vielleicht bemerkt, daß der Methode ein `ActionEvent`-Objekt als Argument beim Aufruf übergeben wird. Dieses Objekt kann dazu verwendet werden, Details über die Komponente zu ermitteln, die das Ereignis erzeugt hat.

`ActionEvent`-Objekt und alle anderen Objekte für Ereignisse sind Teil des Paketes `java.awt.event` und Subklassen der Klasse `EventObject`.

Jeder Methode zur Ereignisbehandlung wird ein Ereignisobjekt irgendeiner Art übergeben. Die Methode `getSource()` des Objekts kann dazu verwendet werden, die Komponente zu ermitteln, die das Ereignis gesendet hat, wie das im folgenden Beispiel gezeigt wird:

```
public void actionPerformed(ActionEvent evt) {
    Object src = evt.getSource();
}
```

Das Objekt, das von der Methode `getSource()` zurückgegeben wird, kann über den Operator `==` mit Komponenten verglichen werden. Die folgenden Anweisungen könnten in dem vorigen `actionPerformed()`-Beispiel verwendet werden:

```
if (src == quitButton)
    quitProgram();
else if (src == sortRecords)
    sortRecords();
```

Dieses Beispiel ruft die Methode `quitProgramm()` auf, wenn das Objekt `quitButton` das Ereignis erzeugt hat, bzw. die Methode `sortRecords()` wird aufgerufen, wenn das Ereignis von der Schaltfläche `sortRecord` stammt.

Viele Methoden zur Ereignisbehandlung werden für die einzelnen Ereignisarten bzw. die verschiedenen Komponenten unterschiedliche Methoden aufrufen. Dadurch ist die Methode zur Ereignisbehandlung leichter zu lesen. Zusätzlich können, wenn sich mehrere Ereignisbehandlungsmethoden in der Klasse befinden, diese dieselben Methoden aufrufen, um bestimmte Aktionen auszuführen.

Eine andere nützliche ../Bilder innerhalb einer Ereignisbehandlungsmethode ist es, das Schlüsselwort `instanceof`

zu verwenden, um zu prüfen, welche Art von Komponente ein Ereignis ausgelöst hat. Das folgende Beispiel könnte in einem Programm mit einer Schaltfläche und einem Textfeld, die beide Aktionsereignisse erzeugen, verwendet werden:

```
void actionPerformed(ActionEvent evt) {
    Object src = evt.getSource();
    if (src instanceof JTextField)
        calculateScore();
    else if (src instanceof JButton)
        quitProgram();
}
```

Das Programm in Listing 21.1 verwendet das Applikationsgerüst aus dem gestrigen Kapitel, um einen JFrame zu erzeugen und Komponenten in diesen einzufügen. Das Programm selbst verfügt über zwei JButton-Komponenten, die verwendet werden, um den Text in der Titelleiste des Frames zu ändern.

Listing 21.1: Der gesamte Quelltext von ChangeTitle.java

```
1: import java.awt.event.*;
2: import javax.swing.*;
3: import java.awt.*;
4:
5: public class ChangeTitle extends JFrame implements ActionListener {
6:     JButton b1 = new JButton("Rosencrantz");
7:     JButton b2 = new JButton("Guildenstern");
8:
9:     public ChangeTitle() {
10:         super("Title Bar");
11:
12:         b1.addActionListener(this);
13:         b2.addActionListener(this);
14:         JPanel pane = new JPanel();
15:         pane.add(b1);
16:         pane.add(b2);
17:
18:         setContentPane(pane);
19:     }
20:
21:     public static void main(String[] args) {
22:         JFrame frame = new ChangeTitle();
23:
24:         WindowListener l = new WindowAdapter() {
25:             public void windowClosing(WindowEvent e) {
26:                 System.exit(0);
27:             }
28:         };
29:         frame.addWindowListener(l);
30:
31:         frame.pack();
32:         frame.setVisible(true);
33:     }
34:
35:     public void actionPerformed(ActionEvent evt) {
36:         Object source = evt.getSource();
37:         if (source == b1)
38:             setTitle("Rosencrantz");
```

```

39:         else if (source == b2)
40:             setTitle("Guildenstern");
41:         repaint();
42:     }
43: }

```

Die Applikation sehen Sie in Abbildung 21.1.



Abbildung 21.1:
Die ChangeTitle-Applikation

Nur 12 Zeilen waren nötig, um in dieser Applikation auf Aktionsereignisse zu reagieren:

- Zeile 1 importiert das Paket `java.awt.event`.
- Die Zeilen 12 und 13 fügen den beiden `JButton`-Objekten Listener für Aktionsereignisse hinzu.
- Die Zeilen 35 bis 42 reagieren auf Aktionsereignisse, die von den beiden `JButton`-Objekten stammen. Mit der Methode `getSource()` des `evt`-Objekts wird die Quelle des Ereignisses ermittelt. Ist dies die Schaltfläche `b1`, wird der Titel des Frames auf `Rosencrantz` gesetzt. Ist dies dagegen die Schaltfläche `b2`, wird dieser auf `Guildenstern` gesetzt. Schließlich muß noch `repaint()` aufgerufen werden, damit die eventuelle Änderung des Titels des Frames ausgegeben wird.

Mit Methoden arbeiten

Der folgende Abschnitt beschreibt die Struktur der einzelnen Methoden zur Ereignisbehandlung und die Methoden, die darin verwendet werden können.

Neben den Methoden, die beschrieben werden, kann die `getSource()`-Methode mit allen Ereignisobjekten verwendet werden, um das Objekt zu ermitteln, das ein Ereignis erzeugt hat.

Aktionsereignisse

Aktionsereignisse treten auf, wenn ein Benutzer eine Aktion mit einer der folgenden Komponenten ausführt: `JButton`, `JCheckBox`, `JComboBox`, `JTextField` oder `JRadioButton`.

Eine Klasse muß die `ActionListener`-Schnittstelle implementieren, um diese Ereignisse zu behandeln. Zusätzlich muß die Methode `addActionListener()` für jede Komponente aufgerufen werden, die ein Aktionsereignis erzeugen soll - es sei denn, Sie wollen die Aktionsereignisse einer Komponente ignorieren.

In der `ActionListener`-Schnittstelle befindet sich lediglich eine Methode: `actionPerformed(ActionEvent)`. Diese hat die folgende Form:

```

public void actionPerformed(ActionEvent evt) {
    // ...
}

```

Zusätzlich zu der Methode `getSource()` können Sie die Methode `getActionCommand()` des `ActionEvent`-Objekts verwenden, um mehr Informationen über die Quelle eines Ereignisses zu erhalten.

Der standardmäßige Rückgabewert dieser Methode (das Action Command der Komponente) ist der Text, der einer Komponente zugewiesen ist, wie z.B. die Beschriftung einer Schaltfläche - einem `JButton`-Objekt. Sie können auch ein anderes Action Command für eine Komponente festlegen, indem Sie deren `setActionCommand(String)`-Methode aufrufen. Das `String`-Argument sollte der Text für das Action Command

sein.

Die folgende Anweisung erzeugt z.B. ein JButton- und ein JTextField-Objekt und weist beiden das Action Command "Sort Files" zu:

```
JButton sort = new JButton("Sort");
JTextField name = new JTextField();
sort.setActionCommand("Sort Files");
name.setActionCommand("Sort Files");
```



Action Commands sind sehr hilfreich, wenn Sie ein Programm schreiben, in dem mehr als eine Komponente dasselbe Verhalten auslösen soll. Ein Beispiel hierfür wäre ein Programm mit einer Schaltfläche Beenden und der Option Beenden in einem Menü. Indem Sie beiden Komponenten dasselbe Action Command geben, können Sie diese mit demselben Code behandeln.

Adjustment-Ereignisse

Adjustment-Ereignisse treten auf, wenn der Wert einer JScrollBar-Komponente mit den Pfeilen der Bildlaufleiste, dem Schieber oder durch Klicks auf die Bildlaufleiste verändert wird. Um diese Ereignisse zu behandeln, muß eine Klasse die Schnittstelle AdjustmentListener implementieren.

In der Schnittstelle AdjustmentListener gibt es nur eine Methode: `adjustmentValueChanged(AdjustmentEvent)`. Diese hat die folgende Form:

```
public void adjustmentValueChanged(AdjustmentEvent evt) {
    // ...
}
```

Um den aktuellen Wert der JScrollBar-Komponente in dieser Ereignisbehandlungsmethode zu ermitteln, rufen Sie die Methode `getValue()` des AdjustmentEvent-Objekts auf. Diese Methode gibt einen Integer zurück, der den Wert der Bildlaufleiste widerspiegelt.

Sie können auch ermitteln, was der Benutzer mit der Bildlaufleiste gemacht hat. Dazu verwenden Sie die Methode `getAdjustmentType()` des AdjustmentEvent-Objekts. Diese gibt einen von fünf Werten zurück. Jeder davon ist eine Klassenvariable der Klasse Adjustment:

- `UNIT_INCREMENT` - Der Wert wurde um 1 erhöht. Dies kann über einen Klick auf die entsprechende Pfeilschaltfläche der Bildlaufleiste oder durch Drücken der entsprechenden Cursor-Tasten erreicht werden.
- `UNIT_DECREMENT` - Der Wert wurde um 1 vermindert.
- `BLOCK_INCREMENT` - Der Wert wurde um einen größeren Wert erhöht. Dies kann durch einen Klick in den Bereich zwischen dem Schieber und der Pfeilschaltfläche erreicht werden.
- `BLOCK_DECREMENT` - Der Wert wurde um einen größeren Wert vermindert.
- `TRACK` - Die Änderung des Wertes wurde durch Bewegen des Schiebers verursacht.

Das Programm in Listing 21.2 illustriert die Verwendung der AdjustmentListener- Schnittstelle. Dem Frame werden eine Bildlaufleiste und ein nichteditierbares Textfeld hinzugefügt. In diesem Feld wird eine Meldung angezeigt, wann immer sich der Wert der Bildlaufleiste verändert.

Listing 21.2: Der gesamte Quelltext von WellAdjusted.java

```
1: import java.awt.event.*;
2: import javax.swing.*;
3: import java.awt.*;
4:
5: public class WellAdjusted extends JFrame implements AdjustmentListener {
```



```

6:      BorderLayout bord = new BorderLayout();
7:      JTextField value = new JTextField();
8:      JScrollBar bar = new JScrollBar(SwingConstants.HORIZONTAL,
9:          50, 10, 0, 100);
10:
11:     public WellAdjusted() {
12:         super("Well Adjusted");
13:
14:         bar.addAdjustmentListener(this);
15:         value.setHorizontalAlignment(SwingConstants.CENTER);
16:         value.setEditable(false);
17:         JPanel pane = new JPanel();
18:         pane.setLayout(bord);
19:         pane.add(value, "South");
20:         pane.add(bar, "Center");
21:
22:         setContentPane(pane);
23:     }
24:
25:     public static void main(String[] args) {
26:         JFrame frame = new WellAdjusted();
27:
28:         WindowListener l = new WindowAdapter() {
29:             public void windowClosing(WindowEvent e) {
30:                 System.exit(0);
31:             }
32:         };
33:         frame.addWindowListener(l);
34:
35:         frame.pack();
36:         frame.setVisible(true);
37:     }
38:
39:     public void adjustmentValueChanged(AdjustmentEvent evt) {
40:         Object source = evt.getSource();
41:         if (source == bar) {
42:             int newValue = bar.getValue();
43:             value.setText("" + newValue);
44:         }
45:         repaint();
46:     }
47: }

```

Abbildung 21.2 zeigt einen Screenshot der Applikation.

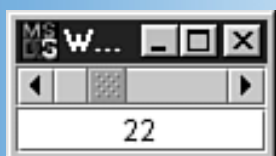


Abbildung 21.2:
Die Ausgabe der WellAdjusted-Applikation



Sie werden sich vielleicht fragen, warum sich in dem Aufruf der Methode `setText()` in Zeile 43 dieses Programms ein Paar leerer Anführungszeichen befindet. Ein solches Paar leerer Anführungszeichen wird als *null-String* bezeichnet. Dieser wird mit dem Integer `newValue` verkettet, um das Argument in einen String zu verwandeln. Java behandelt das Ergebnis immer als String. Wie Sie sich vielleicht erinnern werden, ist dies immer der Fall, wenn ein String und ein Nicht-String miteinander verkettet werden. Der *null-String* ist eine Abkürzung, wenn Sie etwas anzeigen wollen, das kein String ist.

Fokusereignisse

Fokusereignisse treten auf, wenn eine Komponente den Eingabefokus erhält oder verliert. Als Fokus bezeichnet man den Zustand, in dem eine Komponente Tastatureingaben entgegennehmen kann. Wenn eines der Felder den Fokus hat (in einer Benutzerschnittstelle mit mehreren editierbaren Textfeldern), dann blinkt der Cursor in diesem Feld. Jeder Text, der eingegeben wird, richtet sich an diese Komponente.

Der Begriff Fokus bezieht sich auf alle Komponenten, die Eingaben empfangen können. Bei einem `JButton`-Objekt erscheint eine gepunktete Linie um das Objekt herum, wenn dieses den Fokus hat.

Um Fokusereignisse zu behandeln, muß eine Klasse die `FocusListener`-Schnittstelle implementieren. Diese Schnittstelle verfügt über zwei Methoden: `focusGained(FocusEvent)` und `focusLost(FocusEvent)`. Diese haben die folgende Form:

```
public void focusGained(FocusEvent evt) {
    // ...
}

public void focusLost(FocusEvent evt) {
    // ...
}
```

Um festzustellen, welches Objekt den Fokus erhalten bzw. diesen verloren hat, rufen Sie die Methode `getSource()` des `FocusEvent`-Objekts auf, das als Argument an die Methoden `focusGained(FocusEvent)` und `focusLost(FocusEvent)` übergeben wird.

Item-Ereignisse



Item-Ereignisse treten auf, wenn ein Element in einer der folgenden Komponenten gewählt oder abgewählt wird: `JButton`, `JCheckBox`, `JComboBox` oder `JRadioButton`. Eine Klasse muß die Schnittstelle `ItemListener` implementieren, um diese Ereignisse behandeln zu können.

Die Schnittstelle `ItemListener` verfügt nur über eine Methode: `itemStateChanged(ItemEvent)`. Diese hat die folgende Form:

```
void itemStateChanged(ItemEvent evt) {
    // ...
}
```

Um festzustellen, welches Element das Ereignis ausgelöst hat, rufen Sie die Methode `getItem()` des `ItemEvent`-Objekts auf.

Sie können auch ermitteln, ob das Element gewählt oder abgewählt wurde, indem Sie die Methode

getStateChange() aufrufen. Diese Methode gibt einen Integer zurück, der einer der beiden Klassenvariablen ItemEvent.DESELECTED oder ItemEvent.SELECTED entspricht.

Die Anwendung von Item-Ereignissen wird in Listing 21.3 illustriert. Die Applikation SelectItem zeigt die Auswahl, die in einem Kombinationslistenfeld getroffen wird, in einem Textfeld an.

Listing 21.3: Der gesamte Quelltext von SelectItem.java

```

1: import java.awt.event.*;
2: import javax.swing.*;
3: import java.awt.*;
4:
5: public class SelectItem extends JFrame implements ItemListener {
6:     BorderLayout bord = new BorderLayout();
7:     JTextField result = new JTextField(27);
8:     JComboBox pick = new JComboBox();
9:
10:    public SelectItem() {
11:        super("Select Item");
12:
13:        pick.addItemListener(this);
14:        pick.addItem("Navigator");
15:        pick.addItem("Internet Explorer");
16:        pick.addItem("Opera");
17:        pick.setEditable(false);
18:        result.setHorizontalAlignment(SwingConstants.CENTER);
19:        result.setEditable(false);
20:        JPanel pane = new JPanel();
21:        pane.setLayout(bord);
22:        pane.add(result, "South");
23:        pane.add(pick, "Center");
24:
25:        setContentPane(pane);
26:    }
27:
28:    public static void main(String[] args) {
29:        JFrame frame = new SelectItem();
30:
31:        WindowListener l = new WindowAdapter() {
32:            public void windowClosing(WindowEvent e) {
33:                System.exit(0);
34:            }
35:        };
36:        frame.addWindowListener(l);
37:
38:        frame.pack();
39:        frame.setVisible(true);
40:    }
41:
42:    public void itemStateChanged(ItemEvent evt) {
43:        Object source = evt.getSource();
44:        if (source == pick) {
45:            Object newPick = evt.getItem();
46:            result.setText(newPick.toString() + " is the selection.");
47:        }

```

```

48:         repaint();
49:     }
50: }

```

Abbildung 21.3 zeigt diese Applikation, wobei das Element Opera die aktuelle Auswahl in dem Kombinationslistenfeld darstellt. Mit der Methode `toString()` des Objekts, das von der Methode `getItem()` zurückgegeben wird, wird der Text dieses Objekts ermittelt.

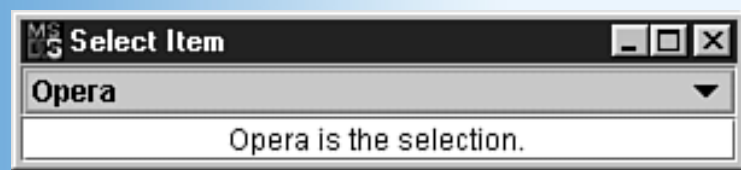


Abbildung 21.3:
Die Ausgabe der SelectItem-Applikation

Tastaturereignisse

Tastaturereignisse treten auf, wenn eine Taste auf der Tastatur gedrückt wird. Jede beliebige Komponente kann diese Ereignisse erzeugen. Eine Klasse muß die Schnittstelle `KeyListener` implementieren, um diese Ereignisse zu unterstützen.

Die `KeyListener`-Schnittstelle verfügt über drei Methoden: `keyPressed(KeyEvent)`, `keyReleased(KeyEvent)` und `keyTyped(KeyEvent)`. Diese haben die folgende Form:

```

public void keyPressed(KeyEvent evt) {
    // ...
}

public void keyReleased(KeyEvent evt) {
    // ...
}

public void keyTyped(KeyEvent evt) {
    // ...
}

```

Die Methode `getKeyChar()` der Klasse `KeyEvent` gibt das Zeichen der Taste zurück, die dieses Ereignis ausgelöst hat. Falls es kein Unicode-Zeichen gibt, das die Taste repräsentiert, gibt die Methode `getKeyChar()` einen Zeichenwert zurück, der der Klassenvariablen `KeyEvent.CHAR_UNDEFINED` entspricht.

Mausereignisse

Mausereignisse werden durch viele verschiedene Arten von Benutzerinteraktionen erzeugt:

- Ein Mausklick
- Die Maus betritt den Bereich einer Komponente
- Die Maus verläßt den Bereich einer Komponente

Jede Komponente kann diese Ereignisse erzeugen. Um sie zu behandeln, muß eine Klasse die Schnittstelle `MouseListener` implementieren. Diese Schnittstelle verfügt über fünf Methoden:

- `mouseClicked(MouseEvent)`
- `mouseEntered(MouseEvent)`
- `mouseExited(MouseEvent)`
- `mousePressed(MouseEvent)`

- `mouseReleased(MouseEvent)`

Dabei entspricht die Form aller dieser Methoden der von `mouseReleased(MouseEvent)` :

```
public void mouseReleased(MouseEvent evt) {
    // ...
}
```

Die folgenden Methoden bietet das `MouseEvent`-Objekt:

- `getClickCount()` - Gibt die Anzahl der Klicks als integer zurück.
- `getPoint()` - Gibt die x,y-Koordinaten eines Mausklicks innerhalb der Komponente als `Point`-Objekt zurück.
- `getX()` - Gibt die x-Koordinate zurück.
- `getY()` - Gibt die y-Koordinate zurück.

MouseMotion-Ereignisse

`MouseMotion`-Ereignisse treten auf, wenn die Maus über eine Komponente bewegt wird. Wie auch die anderen Mausereignisse können diese von jeder Komponente erzeugt werden. Um diese Ereignisse zu unterstützen, muß eine Klasse die Schnittstelle `MouseMotionListener` implementieren.

Die Schnittstelle `MouseMotionListener` verfügt über zwei Methoden: `mouseDragged(MouseMotionEvent)` und `mouseMoved(MouseMotionEvent)`. Diese Methoden haben die folgende Form:

```
public void mouseDragged(MouseEvent evt) {
    // ...
}
```

```
public void mouseMoved(MouseEvent evt) {
    // ...
}
```

Anders als die anderen Event-Listener-Schnittstellen, mit denen Sie es bisher zu tun hatten, hat `MouseMotionListener` keinen eigenen Ereignistyp. Statt dessen werden `MouseEvent`-Objekte verwendet.

Aus diesem Grund können Sie dieselben Methoden wie bei Mausereignissen verwenden: `getClick()`, `getPoint()`, `getX()` und `getY()`.

Fensterereignisse

Fensterereignisse treten auf, wenn ein Benutzer Fenster öffnet oder schließt. Unter dem Begriff Fenster werden Objekte wie z.B. `JFrame` oder `JWindow` verstanden. Eine Klasse muß die Schnittstelle `WindowListener` implementieren, um diese Ereignisse zu unterstützen.

Die `WindowListener`-Schnittstelle verfügt über sieben Methoden:

- `windowActivated(WindowEvent)`
- `windowClosed(WindowEvent)`
- `windowClosing(WindowEvent)`
- `windowDeactivated(WindowEvent)`
- `windowDeiconified(WindowEvent)`
- `windowIconified(WindowEvent)`
- `windowOpened(WindowEvent)`

Alle haben sie dieselbe Form wie die Methode `windowOpened(WindowEvent)`:

```
public void windowOpened(WindowEvent evt) {
```



```
// ...
}
```

Die Methoden `windowClosing()` und `windowClosed()` sind sich ähnlich, allerdings wird erstere aufgerufen, wenn sich das Fenster schließt, und die zweite, nachdem das Fenster geschlossen ist. Sie können in der Methode `windowClosing()` das Schließen des Fensters verhindern.

Ein Beispiel: die `SwingColorTest`-Applikation

Gestern haben Sie eine grafische Benutzeroberfläche für das RGB-zu-HSB-Konvertierungs-Programm mit Swing-Komponenten erstellt.

Um mehr Erfahrung im Umgang mit dem Ereignisbehandlungsmodell von Swing zu bekommen, verwandeln Sie die Benutzerschnittstelle von `SwingColorTest` in ein funktionierendes Programm.

Gestern haben Sie zwei Klassen für dieses Projekt erzeugt:

`SwingColorTest` und `SwingColorControls`.

`SwingColorTest` beinhaltet das Hauptfenster der Applikation und die `main()`-Methode, mit der das Fenster eingerichtet wird. `SwingColorControls`, eine Hilfsklasse, ist ein Panel, das drei Labels und drei Textfelder beinhaltet, die zur Wahl der Farbe verwendet werden.

Sämtliche Benutzereingaben finden in den Objekten der Klasse `SwingColorControls` statt - über die Textfelder werden sRGB- oder HSB-Werte definiert.

Aus diesem Grund wird das gesamte Verhalten für die Ereignisbehandlung in die Klasse `SwingColorControls` eingefügt.

Als erstes müssen Sie dafür sorgen, daß die Klasse `SwingColorControls` zwei Arten von Ereignissen behandelt: Aktionsereignisse und Fokusereignisse. Mit der Anweisung `implements` müssen in der Klassendeklaration die Schnittstellen `ActionListener` und `FocusListener` implementiert werden. Dies ist im folgenden gezeigt:

```
class SwingColorControls extends JPanel
    implements ActionListener, FocusListener {
```

Als nächstes müssen die entsprechenden Listener den Textfeldern (`tfield1`, `tfield2` und `tfield3`) in der Klasse hinzugefügt werden. Diese Listener müssen Sie hinzufügen, nachdem die Textfelder erzeugt wurden und bevor diese in einen Container eingefügt werden. Dazu können Sie die folgenden Anweisungen verwenden:

```
tfield1.addFocusListener(this);
tfield2.addFocusListener(this);
tfield3.addFocusListener(this);
tfield1.addActionListener(this);
tfield2.addActionListener(this);
tfield3.addActionListener(this);
```

Schließlich müssen Sie noch alle Methoden, die in den beiden Schnittstellen, die diese Klasse implementiert, definiert sind, hinzufügen:

`actionPerformed(ActionEvent)`, `focusLost(FocusEvent)` und `focusGained(FocusEvent)` .

Die `SwingColorControls`-Objekte werden zur Eingabe von numerischen Werten für eine Farbe verwendet. Die Eingabe von Werten sorgt dafür, daß die Farbe auf einem Panel angezeigt wird. Außerdem wird das zweite `SwingColorControls`-Objekt so aktualisiert, daß sich die Änderung der Farbe in den darin befindlichen Zahlenwerten widerspiegelt.

Es gibt zwei Möglichkeiten für den Benutzer, die Auswahl einer neuen Farbe abzuschließen - durch Drücken der Eingabetaste innerhalb eines Textfeldes, wodurch ein Aktionsereignis ausgelöst wird, und durch Verlassen des Feldes, um den Wert eines anderen Feldes zu editieren, was zu einem Fokusereignis führt.

Die folgenden Anweisungen bilden die actionPerformed()- und focusLost()-Methoden, die Sie in Ihrer Klasse einfügen sollten:

```
public void actionPerformed(ActionEvent evt) {
    if (evt.getSource() instanceof TextField)
        frame.update(this);
}
public void focusLost(FocusEvent evt) {
    frame.update(this);
}
```

Da das Ereignis, das bei Erhalt des Fokus erzeugt wird, hier nicht behandelt wird, die dazugehörige Methode (focusGained()) der Schnittstelle aber implementiert werden muß, sollten Sie eine leere Methode einfügen:

```
public void focusGained(FocusEvent evt) { }
```

Die Ereignisbehandlungsmethoden, die in SwingColorControls eingefügt wurden, rufen eine Methode in der Klasse SwingColorTest auf (update(SwingColorControls)).

Diese Methode beinhaltet keinerlei Verhaltensweisen zur Ereignisbehandlung - Sie aktualisiert das Farbmuster und die SwingColorControls-Objekte, damit diese die Farbänderung widerspiegeln. Dies entspricht der Version, die Sie am Tag 14 erstellt haben.

Listing 21.4 beinhaltet die Applikation mit den beiden Klassen SwingColorTest und SwingColorControls.

Listing 21.4: Der gesamte Quelltext von SwingColorTest.java

```
1: import java.awt.*;
2: import java.awt.event.*;
3: import javax.swing.*;
4:
5: public class SwingColorTest extends JFrame {
6:     SwingColorControls RGBcontrols, HSBcontrols;
7:     JPanel swatch;
8:
9:     public SwingColorTest() {
10:         super("Color Test");
11:
12:         JPanel pane = new JPanel();
13:         pane.setLayout(new GridLayout(1, 3, 5, 15));
14:         swatch = new JPanel();
15:         swatch.setBackground(Color.black);
16:         RGBcontrols = new SwingColorControls(this, "Red",
17:             "Green", "Blue");
18:         HSBcontrols = new SwingColorControls(this, "Hue",
19:             "Saturation", "Brightness");
20:         pane.add(swatch);
21:         pane.add(RGBcontrols);
22:         pane.add(HSBcontrols);
23:
24:         setContentPane(pane);
25:     }
26:
27:     public static void main(String[] args) {
28:         JFrame frame = new SwingColorTest();
29:
30:         WindowListener l = new WindowAdapter() {
31:             public void windowClosing(WindowEvent e) {
```

```

32:         System.exit(0);
33:     }
34: };
35: frame.addWindowListener(l);
36:
37: frame.pack();
38: frame.setVisible(true);
39: }
40:
41: public Insets getInsets() {
42:     return new Insets(10, 10, 10, 10);
43: }
44:
45: void update(SwingColorControls controlPanel) {
46:     Color c;
47:     // Die String-Werte aus den Textfeldern auslesen und in ints
48:     // konvertieren
49:     int value1 = Integer.parseInt(controlPanel.tfield1.getText());
50:     int value2 = Integer.parseInt(controlPanel.tfield2.getText());
51:     int value3 = Integer.parseInt(controlPanel.tfield3.getText());
52:
53:     if (controlPanel == RGBcontrols) {
54:         // RGB geändert, HSB aktualisieren
55:         c = new Color(value1, value2, value3);
56:
57:         // RGB-Werte in HSB-Werte konvertieren
58:         float[] HSB = Color.RGBtoHSB(value1, value2, value3,
59:             (new float[3]));
60:         HSB[0] *= 360;
61:         HSB[1] *= 100;
62:         HSB[2] *= 100;
63:
64:         // HSB-Felder zurücksetzen
65:         HSBcontrols.tfield1.setText(String.valueOf((int)HSB[0]));
66:         HSBcontrols.tfield2.setText(String.valueOf((int)HSB[1]));
67:         HSBcontrols.tfield3.setText(String.valueOf((int)HSB[2]));
68:     } else {
69:         // HSB geändert, RGB aktualisieren
70:         c = Color.getHSBColor((float)value1 / 360,
71:             (float)value2 / 100, (float)value3 / 100);
72:
73:         // RGB-Felder zurücksetzen
74:         RGBcontrols.tfield1.setText(String.valueOf(c.getRed()));
75:         RGBcontrols.tfield2.setText(String.valueOf(c.getGreen()));
76:         RGBcontrols.tfield3.setText(String.valueOf(c.getBlue()));
77:     }
78:
79:     // Farbanzeiger aktualisieren
80:     swatch.setBackground(c);
81:     swatch.repaint();
82: }
83: }
84:

```

```

85: class SwingColorControls extends JPanel
86:     implements ActionListener, FocusListener {
87:
88:     SwingColorTest frame;
89:     JTextField tfield1, tfield2, tfield3;
90:
91:     SwingColorControls(SwingColorTest parent,
92:         String l1, String l2, String l3) {
93:
94:         frame = parent;
95:         setLayout(new GridLayout(3,2,10,10));
96:         tfield1 = new JTextField("0");
97:         tfield2 = new JTextField("0");
98:         tfield3 = new JTextField("0");
99:         tfield1.addFocusListener(this);
100:        tfield2.addFocusListener(this);
101:        tfield3.addFocusListener(this);
102:        tfield1.addActionListener(this);
103:        tfield2.addActionListener(this);
104:        tfield3.addActionListener(this);
105:        add(new JLabel(l1, JLabel.RIGHT));
106:        add(tfield1);
107:        add(new JLabel(l2, JLabel.RIGHT));
108:        add(tfield2);
109:        add(new JLabel(l3, JLabel.RIGHT));
110:        add(tfield3);
111:    }
112:
113:    public Insets getInsets() {
114:        return new Insets(10, 10, 0, 0);
115:    }
116:
117:    public void actionPerformed(ActionEvent evt) {
118:        if (evt.getSource() instanceof TextField)
119:            frame.update(this);
120:    }
121:
122:    public void focusLost(FocusEvent evt) {
123:        frame.update(this);
124:    }
125:
126:    public void focusGained(FocusEvent evt) { }
127: }

```

Abbildung 21.4 zeigt das fertige Produkt.

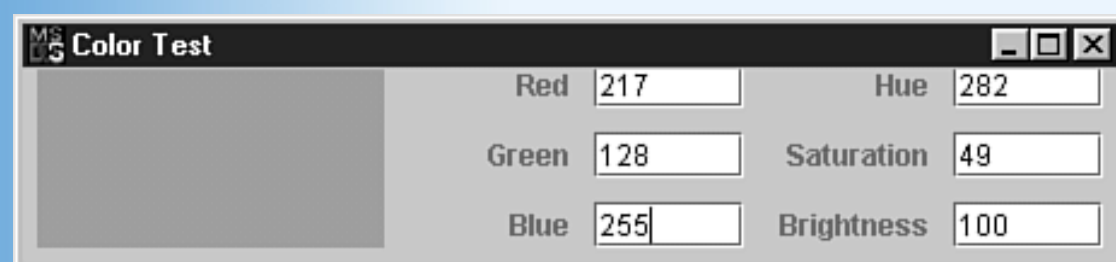


Abbildung 21.4:

Die SwingColorTest-Applikation

Zusammenfassung

Intern ist das Ereignisbehandlungssystem von Swing wesentlich robuster und einfacher zu erweitern, um neue Arten der Interaktion mit dem Benutzer zu verarbeiten.

Nach außen sollte das System auch vom Standpunkt der Programmierung her mehr Sinn machen. Die Ereignisbehandlung wird einem Programm immer über dieselben Schritte hinzugefügt:

- Der Klasse, die die Methoden zur Ereignisbehandlung enthalten wird, wird eine Listener-Schnittstelle hinzugefügt.
- Jede Komponente, die ein Ereignis erzeugen soll, erhält einen Listener zugeteilt.
- Die Methoden werden eingefügt. Jede erwartet als einziges Argument ein Objekt einer Subklasse der Klasse EventObject.
- Die Methoden des Objekts der Subklasse von EventObject, wie z.B. getSource(), werden dazu verwendet, mehr über die Komponente, die das Ereignis erzeugt hat, und die Art des aufgetretenen Ereignisses zu erfahren.

Sobald Sie diese Schritte kennen, können Sie mit jeder der verschiedenen Listener- Schnittstellen und Ereignis-Klassen arbeiten. Außerdem können Sie den Umgang mit neuen Listenern leicht erlernen, sobald diese in Swing hinzugefügt werden.

Dies bringt uns zu dem Hauptereignis: Der Abschluß der 21tägigen Reise durch die Sprache Java. Jetzt, da Sie die Gelegenheit hatten, mit der Syntax und den Kernklassen, die Java ausmachen, zu arbeiten, sind Sie bereit, die wirklich harten Nüsse zu knacken: Ihre eigenen Programme.

Herzlichen Glückwunsch! Nachdem Sie nun eine Einführung in die wohl bemerkenswerteste Programmiersprache des letzten Jahrzehnts erhalten haben, ist es an Ihnen, bemerkenswerte Dinge damit zu tun.

Während Sie viel Zeit in Ihre eigenen Programme investieren, neue Features lernen und die Java-Klassen in Ihren eigenen Paketen erweitern, werden Sie einen weiteren Grund kennenlernen, warum der Name Java gut gewählt war:

Java kann, wie sein koffeinhaltiges Gegenstück, abhängig machen.

»Laß' es nicht so enden. Sag' ihnen, daß ich etwas gesagt habe.«

- Die letzten Wort von Pancho Villa (1877-1923)

Fragen und Antworten

Frage:

Kann man das Verhalten zur Ereignisbehandlung eines Programmes nicht in eine eigene Klasse auslagern, anstelle diese in den Code für die Benutzerschnittstelle einzufügen?

Frage:

Man kann, und viele Programmierer werden Ihnen sagen, daß es eine gute Methode ist, wenn Sie Ihre Programme so entwerfen. Durch die Trennung des Entwurfs der Benutzerschnittstelle von dem Code zur Ereignisbehandlung wird es möglich, diese beiden Bereiche getrennt voneinander zu entwickeln - die SwingColorTest-Applikation von heute zeigt den gegenteiligen Ansatz. Dies vereinfacht die Pflege des Projektes, da zusammengehörige Verhaltensweisen gruppiert und von Verhaltensweisen, die damit nicht im Zusammenhang stehen, isoliert sind.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).

Tag A

Java- Sprachübersicht

Dieser Anhang enthält eine Übersicht über die Sprache Java und dient als Nachschlageteil für dieses Buch.



Das ist keine Übersicht der Syntax oder der Techniken der Sprache. Vielmehr dient der Teil zum Nachschlagen von Informationen zu Themen, die Sie in den Lektionen dieses Buches gelernt haben. Wenn Sie eine technische Beschreibung der Sprache benötigen, besuchen Sie am besten die JavaSoft-Website (<http://java.sun.com/docs>) und laden die Spezifikation herunter. Diese enthält auch eine vollständige Beschreibung der Syntax.

Schlüsselwörter und Symbole sind in Schreibmaschinenschrift angegeben. Argumente und andere zu ersetzende Teile sind in kursiver Schreibmaschinenschrift abgesetzt.

Optionen stehen zwischen Klammern (abgesehen vom Array-Abschnitt). Sind mehrere Optionen verfügbar, die sich gegenseitig ausschließen, werden sie durch ein Pipe- Zeichen (|) getrennt, z.B.:

[public | private | protected] Typ Varname

Reservierte Wörter

Die folgenden Wörter sind für die Java-Sprache reserviert (die teilweise im derzeitigen Release noch nicht benutzt werden). Sie können diese Wörter nicht für Namen von Klassen, Methoden oder Variablen verwenden.

abstract	do	import	public	try
boolean	double	instanceof	return	void
break	else	int	short	volatile
byte	extends	interface	static	while
case	final	long	super	
catch	finally	native	switch	
char	float	new	synchronized	
class	for	null	this	
const	goto	package	throw	
continue	if	private	throws	
default	implements	protected	transient	

Die Schlüsselwörter true, false und null sind ebenfalls reserviert - allerdings für die Verwendung als Literale.

Kommentare

```
/* Das ist ein mehrzeiliger Kommentar - der Kommentartext
kann mehrere Zeilen umfassen*/
// Das ist ein einzeiliger Kommentar
/** Das ist ein Javadoc-Kommentar */
```

Literale

<i>Zahl</i>	Typ int
<i>Zahl</i> [l] [L]	Typ long
<i>0xhex</i>	Hex Integer
<i>0Xhex</i>	Hex Integer
<i>0octal</i>	Oktal Integer
[<i>Zahl</i>]. <i>Zahl</i>	Typ double
<i>Zahl</i> [f f]	Typ float
<i>Zahl</i> [d D]	Typ double
[+ -] <i>Zahl</i>	Vorzeichen
<i>ZahleZahl</i>	Exponent
<i>ZahlEZahl</i>	Exponent
'einZeichen'	Einzelnes Zeichen
"mehrereZeichen"	Zeichenkette
""	Leere Zeichenkette
\b	Rückschritt
\t	Tabulator
\n	Zeilenvorschub
\f	Seitenvorschub
\r	Wagenrücklauf
\"	Doppeltes Anführungszeichen
\'	Einfaches Anführungszeichen
\\	Backslash
\uNNNN	Unicode-Zeichenersatz (NNNN ist hex)
true	Boolesch
false	Boolesch

Variablendeklaration

[byte short int long] Varname	Ganzzahlen (einen Typ auswählen)
[float double] Varname	Gleitpunktzahlen (einen Typ auswählen)
char Varname;	Zeichen
boolean Varname;	Boolesch
Klassenname Varname;	Klassentypen
Typ Varname, Varname, Varname;	Mehrere Variablen

Die folgenden Optionen sind nur für Klassen- und Instanzvariablen verfügbar und können in einer Variablendeklaration benutzt werden:

[static] Variablendeklaration	Klassenvariablen
[final] Variablendeklaration	Konstanten
[public private protected] Variablendeklaration	Zugriffskontrolle

Wertzuweisung an Variablen

Variable = Wert	Zuweisung
Variable++	Postfix-Inkrement
++Variable	Präfix-Inkrement
Variable-	Postfix-Dekrement
-Variable	Präfix-Dekrement
Variable += Wert	Addieren und Zuweisen
Variable -= Wert	Subtrahieren und Zuweisen
Variable *= Wert	Multiplizieren und Zuweisen
Variable = Wert	Dividieren und Zuweisen
Variable %= Wert	Modulo und Zuweisen
Variable &= Wert	UND und Zuweisen
Variable = Wert	ODER und Zuweisen
Variable ^= Wert	XOR und Zuweisen
Variable <<= Wert	Links-Shift und Zuweisen
Variable >>= Wert	Rechts-Shift und Zuweisen
Variable >>>= Wert	Mit Nullen füllender Rechts-Shift und zuweisen

Operatoren

Arg + Arg	Addition
Arg - Arg	Subtraktion
Arg * Arg	Multiplikation
Arg / Arg	Division
Arg % Arg	Modulo
Arg < Arg	Kleiner als
Arg > Arg	Größer als
Arg <= Arg	Kleiner als oder gleich
Arg >= Arg	Größer als oder gleich
Arg == Arg	Gleich
Arg != Arg	Nicht gleich
Arg && Arg	Logisches UND
Arg Arg	Logisches ODER
! Arg	Logisches NICHT
Arg & Arg	UND
Arg Arg	ODER
Arg ^ Arg	XOR
Arg << Arg	Links-Shift
Arg >> Arg	Rechts-Shift
Arg >>> Arg	Mit Nullen füllender Rechts-Shift
~ Arg	Komplement
(Typ)Ausdruck	Umwandlung (Casting)
Arg instanceof Klasse	Instanz von
Test ? AnweisungFürTrue: AnweisungFürFalse	Ternärer (if)-Operator

Objekte

new Klasse();	Neue Instanz erstellen
new Klasse(Arg,Arg,Arg...)	Neue Instanz mit Argumenten
new Typ(Arg,Arg,Arg...)	Neue Instanz einer anonymen Klasse
primary.new Typ(Arg,Arg,Arg...)	Neue Instanz einer anonymen Klasse
Objekt.Variable	Instanzvariable
Objekt.Klassenvariable	Klassenvariable

Klasse.Klassenvariable	Klassenvariable
Objekt.Methode()	Instanzmethode (ohne Argumente)
Objekt.Methode(Arg,Arg,Arg...)	Instanzmethode (mit Argumenten)
Objekt.Klassenmethode()	Klassenmethode (ohne Argumente)
Objekt.Klassenmethode(Arg,Arg,Arg...)	Klassenmethode (mit Argumenten)
Klasse.Klassenmethode()	Klassenmethode (ohne Argumente)
Klasse.Klassenmethode(Arg,Arg,Arg...)	Klassenmethode (mit Argumenten)

Arrays



Die Klammern in diesem Abschnitt sind Teil des Arrays bzw. der Zugriffsanweisungen und stehen nicht für Optionen wie in den vorherigen Abschnitten.

Typ Varname[]	Array-Variable
Typ[] Varname	Array-Variable
new Typ[AnzElemente]	Neues Array-Objekt
new Typ[] {Initialisierung}	Neues anonymes Array-Objekt
Array[Index]	Elementzugriff
Array.length	Array-Länge

Schleifen und Bedingungen

if (Test) Block	Bedingung
if (Test) Block else Block	Bedingung mit else
switch (Test) { case Wert : Anweisungen case Wert : Anweisungen ... default : Anweisungen }	switch (nur für int- und char-Typen)
for (Initialisierung; Test; Veränderung)	Block mit for-Schleife
while (Test) Block	while-Schleife
do Block while (Test)	do-Schleife

break [Marke]	Bricht eine Schleife oder switch-Anweisung ab
continue [Marke]	Setzt die Schleife mit der nächsten Iteration fort
Marke:	Benannte Schleife

Klassendefinitionen

class Klassenname Block	Einfache Klassendefinition

Die folgenden optionalen Modifier können in die Klassendefinition eingefügt werden:

[final] class Klassenname Block	Klasse kann nicht abgeleitet werden
[abstract] class Klassenname Block	Klasse kann nicht instanziiert werden
[public] class Klassenname Block	Zugriff außerhalb des Pakets möglich
class Klassenname [extends Superklasse] Block	Superklasse ableiten
class Klassenname [implements Schnittstellen] Block	Eine oder mehrere Schnittstellen implementieren

Methoden- und Konstruktor-Definitionen

Die Basismethode sieht wie folgt aus, wobei Rückgabotyp ein Typname, ein Klassenname oder void ist:

Rückgabotyp Methodenname() Block	Basismethode
Rückgabotyp Methodenname(Parameter, Parameter, ...) Block	Methode mit Parametern

Methodenparameter sehen so aus:

Typ Parametername

Methodenvariationen können eines der folgenden optionalen Schlüsselwörter enthalten:

[abstract] Rückgabotyp MethodenName() Block	Abstrakte Methode
[static] Rückgabotyp MethodenName() Block	Klassenmethode
[native] Rückgabotyp MethodenName() Block	Native Methode
[final] Rückgabotyp MethodenName() Block	final-Methode (Methode kann nicht überschrieben werden)
[synchronized] Rückgabotyp MethodenName() Block	Thread-Sperre vor Ausführung
[public private protected] Rückgabotyp Methodenname()	Zugriffskontrolle

Konstruktoren sehen so aus:

Klassenname() Block	Basis-Konstruktor
Klassenname(Parameter, Parameter, Parameter...) Block	Konstruktor mit Parametern
[public private protected] Klassenname() Block	Zugriffskontrolle

Im Rumpf von Methoden bzw. Konstruktoren können folgende Referenzen und Methoden verwendet werden:

this	Verweist auf das aktuelle Objekt
Klassenname.this	Verweist auf ein bestimmtes inneres Klassenobjekt
super	Verweist auf eine Superklasse
super.Methodenname()	Ruft die Methode einer Superklasse auf
this(...)	Ruft den Konstruktor einer Klasse auf
super(...)	Ruft den Konstruktor einer Superklasse auf
Typ.class	Gibt ein Klassenobjekt für den Typ zurück
return [Wert]	Gibt einen Wert zurück

Import

Import Paket.Klassenname	Importiert eine spezifische Klasse
Import Paket.*	Importiert alle Klassen des Pakets
Package Paketname	Die Klassen dieser Datei gehören zu diesem Paket
Interface Schnittstellename	Einfache Schnittstellendefinition
Interface Schnittstellename extends AndereSchnittstelle Block	Erweitert eine bestehende Schnittstelle
public interface Schnittstellename Block	Erzeugt eine öffentliche Schnittstelle
abstract interface Schnittstellename Block	Erzeugt eine abstrakte Schnittstelle

Überwachung

synchronized (Objekt) Block	Schützt das Objekt vor dem gleichzeitigen Zugriff durch mehrere Threads
try Block	Geschützte Anweisungen
catch (Exception) Block	Wird im Fall von Exception ausgeführt

[finally Block]	Wird immer ausgeführt
try Block [catch (Exception) Block] finally Block	Wie im vorherigen Beispiel (kann wahlweise catch oder finally, jedoch nicht beides benutzen)

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Tag B

Die Java-Klassenbibliothek

In diesem Anhang finden Sie eine allgemeine Übersicht über die in den Java-Standardpaketen verfügbaren Klassen (d.h. die Klassen, die garantiert in jeder Java-Implementierung verfügbar sind). Dieser Anhang dient lediglich als allgemeiner Nachschlageteil. Die folgenden Pakete der Klassenbibliothek werden behandelt. Speziellere Informationen über die einzelnen Klassen (wie z.B. Vererbung, Variablen und Methoden) finden Sie im Bereich Products & APIs der JavaSoft-Website (<http://java.sun.com>).

Einige Klassen, Methoden und Variablen werden als deprecated (verworfen) aufgeführt. Diese sind im Java 1.2 API aus Gründen der Kompatibilität mit früheren Versionen enthalten, wurden allerdings durch verbesserte Versionen ersetzt. Details zu den Ersetzungen finden Sie in der API-Dokumentation. Die folgenden Pakete werden in diesem Anhang beschrieben:

- java.applet
- java.awt
- java.awt.color
- java.awt.datatransfer
- java.awt.dnd
- java.awt.event
- java.awt.font
- java.awt.geom
- java.awt.im
- java.awt.image
- java.awt.image.renderable
- java.awt.print
- java.beans
- java.beans.beancontext
- java.io
- java.lang
- java.lang.ref
- java.lang.reflect
- java.math
- java.net
- java.rmi
- java.rmi.activation
- java.rmi.dgc
- java.rmi.registry
- java.rmi.server

- java.security
- java.security.acl
- java.security.cert
- java.security.interfaces
- java.security.spec
- java.sql
- java.text
- java.util
- java.util.jar
- java.util.mime (nur Java 1.1)
- java.util.zip
- javax.accessibility
- javax.swing
- javax.swing.border
- javax.swing.colorchooser
- javax.swing.event
- javax.swing.filechooser
- javax.swing.plaf
- javax.swing.plaf.basic
- javax.swing.plaf.metal
- javax.swing.plaf.multi
- javax.swing.table
- javax.swing.text
- javax.swing.text.html
- javax.swing.tree
- javax.swing.undo

java.applet

Dieses Paket ermöglicht es einem Java-Programm, als Teil einer Webseite ausgeführt zu werden und Sounddateien abzuspielen. Java 1.2 fügt dem noch verbesserte Audio- Fähigkeiten hinzu: Unterstützung von 8-Bit- und 16-Bit-Dateien, Samplingraten von 8 kHz bis 48 kHz und die folgenden Formate: AIFF, AU, RMF, WAV und MIDI Typ 0 und Typ 1. Eine neue Methode in der Klasse AudioClip-Klasse, `newAudioClip()`, ermöglicht es nun auch Applikationen, Sounddateien abzuspielen.

Schnittstellen

AppletContext	Methoden, die auf einen Applet-Kontext verweisen
AppletStub	Methoden für die Implementierung von Appletviewern
AudioClip	Methoden zum Abspielen von Audiodateien

Klassen

Applet	Die Applet-Basisklasse

java.awt

Das Paket java.awt enthält Klassen und Schnittstellen, die das Abstract Windowing Toolkit bilden. Diese dienen der Grafikverarbeitung und dem Aufbau grafischer Benutzerschnittstellen.

Schnittstellen

ActiveEvent	Methoden für Ereignisse, die sich selbst senden können (Java 1.2)
Adjustable	Methoden für Objekte mit justierbaren numerischen Werten innerhalb eines Wertebereiches (Java 1.1)
Composite	Methoden, um ein benutzerdefiniertes Composit-Bild eines neuen 2D-Bildes mit dem darunterliegenden Grafikbereich zu erstellen (Java 1.2)
CompositeContext	Methoden, um die Umgebung für eine benutzerdefinierte 2D-Grafik-Composit-Operation zu definieren (Java 1.2)
EventSource	Methoden für ereigniserzeugende Objekte (in der Java-1.1-Beta-Version enthalten; im Final-Release von Java 1.1 und in Java 1.2 nicht enthalten)
ItemSelectable	Methoden für Objekte, die eine Gruppe von auswählbaren Elementen enthalten (Java 1.1)
LayoutManager	Methoden für das Container-Layout
LayoutManager2	Minimale Repräsentation von Methoden, um Komponenten auf Basis von Constraints-Objekten, die genau festlegen wo sich eine Komponente in einem Container befinden soll zu platzieren. (Java 1.1)
MenuContainer	Methoden für menübezogene Container (die Methode postEvent() ist in Java 1.2 als deprecated deklariert)
Paint	Methoden zur Definition von Mustern für 2D-Zeichenoperationen (Java 1.2)
PaintContext	Methoden zur Definition der Umgebung einer 2D-Zeichenoperation mit einem benutzerdefinierten Muster (Java 1.2)
PrintGraphics	Methoden zur Erstellung eines Grafikkontextes zum Drucken (Java 1.1)
Shape	Methoden für geometrische Formen (eingeführt mit Java 1.1 und als Teil der 2D-Funktionalität von Java 1.2 überarbeitet)
Stroke	Methoden zur Repräsentation des Bereiches des Striches in 2D-Grafikoperationen (Java 1.2)
Transparency	Methoden zur Definition von Transparenzmodi (Java 1.2)

Klassen

AlphaComposite	Der Alpha-Kanal für die Bildverarbeitung (Java 1.2)
AWTEvent	Die »Mutter« aller AWT-Ereignisse (Java 1.1)
AWTEventMulticaster	Multicast-Ereignisverteiler (Java 1.1)
AWTPermission	Ein Weg, um Berechtigungen zu erzeugen, die den Zugriff auf AWT-Ressourcen kontrollieren (Java 1.2)
BasicStroke	Ein Zeichenstrich, der eine Stiftform und die Form an den Linienenden repräsentiert (Java 1.2)
BorderLayout	Layout-Manager für Anordnung von Elementen in einem rechteckigen Bereich in der Mitte und in rechteckigen Bereichen der vier Himmelsrichtungen (die Methode addLayoutComponent() wurde seit Java 1.1 als deprecated deklariert)
Button	Schaltfläche der Benutzeroberfläche
Canvas	Zeichenbereich zum Zeichnen und Ausführen anderer Grafikoperationen
CardLayout	Layout-Manager, bei dem Komponenten wie in einem Kartenstapel verwaltet werden, von dem immer nur die oberste Karte sichtbar ist (die Methode addLayoutComponent() wurde seit Java 1.1 als deprecated deklariert)
Checkbox	Kontrollfeld
CheckboxGroup	Gruppe sich ausschließender Kontrollfelder (Optionsschaltflächen) (die Methoden setCurrent() und getCurrent() wurde seit Java 1.1 als deprecated deklariert)
CheckboxMenuItem	Umschaltende Menüoption
Choice	Pop-up-Auswahlmenü (die Methode countItems() wurde seit Java 1.1 als deprecated deklariert)
Color	Abstrakte Darstellung einer Farbe
Component	Abstrakte allgemeine Klasse für alle Komponenten der Benutzeroberfläche (die folgenden Methoden wurden seit Java 1.1 als deprecated deklariert: action(), bounds(), deliverEvent(), disable(), enable(), getPeer(), gotFocus(), handleEvent(), hide(), inside(), keyDown(), keyUp(), layout(), locate(), location(), lostFocus(), minimumSize(), mouseDown(), mouseDrag(), mouseEnter(), mouseExit(), mouseMove(), mouseUp(), move(), nextFocus(), postEvent(), preferredSize(), reshape(), resize(), show() und size())
Component-Orientation	Bestimmt abhängig von der Sprache die Orientierung von Text (Java 1.2)
Container	Abstrakte Eigenschaft einer Komponenten, die andere Komponenten oder Container enthalten kann (die folgenden Methoden wurden seit Java 1.1 als deprecated deklariert: countComponents(), deliverEvent(), insets(), layout(), locate(), minimumSize() und preferredSize())
Cursor	Bildschirmcursor (Java 1.1)

Dialog	Fenster für einfache Interaktionen mit dem Benutzer
Dimension	Objekt, das Höhe und Breite einer Komponente oder anderer Objekte repräsentiert
Event	Objekt, das im Ereignisbehandlungsmodell von Java 1.0.2 Ereignisse repräsentiert, die vom System oder von Benutzereingaben herrühren (in Java 1.1 und Java 1.2 durch neues Modell abgelöst, aber zur Abwärtskompatibilität weiterhin vorhanden)
EventQueue	Warteschlange von zur Verarbeitung anstehenden Ereignissen (Java 1.1)
FileDialog	Dialogfenster zum Auswählen von Dateinamen vom lokalen Dateisystem
FlowLayout	Layout-Manager, der Objekte von links nach rechts in Reihen anlegt
Font	Abstrakte Darstellung eines Fonts
FontMetrics	Meßwerte zur Darstellung eines Fonts am Bildschirm (die Methode getMaxDecent() wurde seit Java 1.1 als deprecated deklariert und durch die Methode getMaxDescent() ersetzt)
Frame	Toplevel-Fenster mit Titel (die Methoden getCursorType() und setCursor() wurden seit Java 1.1 als deprecated deklariert; die folgenden Variablen sind seit Java 1.2 als deprecated deklariert: CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR und WAIT_CURSOR)
GradientPaint	Dient zur Definition einer Verlaufsfüllung in einer Form (Java 1.2)
Graphics	Abstrakte Klasse zur Darstellung eines Grafikkontextes und zum Zeichnen von Formen und Objekten (Die Methode getClipRect() wurde seit Java 1.1 als deprecated deklariert)
Graphics2D	Eine Erweiterung von Graphics, die ausgefeilte 2-D-Darstellungen ermöglicht (Java 1.2)
GraphicsConfig-Template	Template für die Erstellung von GraphicsConfiguration-Objekten (Java 1.2)
GraphicsConfiguration	Die physikalischen Charakteristiken eines grafischen Ausgabegerätes (Java 1.2)
GraphicsDevice	Ein grafisches Ausgabegerät, wie z.B. ein Drucker oder ein Monitor
GraphicsEnvironment	Eine Repräsentation aller grafischen Ausgabegeräte und Font-Objekte, die in einer Umgebung präsent sind (Java 1.2)
GridBagConstraints	Constraints für mit GridBagLayout angelegte Komponenten
GridBagLayout	Layout-Manager, der Komponenten horizontal und vertikal auf der Basis von GridBagConstraints auslegt
GridLayout	Layout-Manager mit Zeilen und Spalten; Elemente werden in die einzelnen Zellen des Rasters eingefügt

Image	Abstrakte Darstellung eines Bitmap-Bildes
Insets	Abstände vom äußeren Rand des Fensters; Verwendung beim Layout von Komponenten
Label	Beschriftungstext für Komponenten der Benutzeroberfläche
List	Listefeld (Die folgenden Methoden wurden seit Java 1.1 als deprecated deklariert: <code>allowsMultipleSelections()</code> , <code>clear()</code> , <code>countItems()</code> , <code>delItems()</code> , <code>isSelected()</code> , <code>minimumSize()</code> , <code>preferredSize()</code> , <code>setMultipleSelections()</code> , <code>addItem()</code> und <code>delItem()</code> .)
MediaTracker	Möglichkeit zur Verfolgung des Status von Medienobjekten, die über das Netz geladen werden
Menu	Menü, das Menüoptionen enthält und ein Container in einer Menüleiste ist (die Methode <code>countItems()</code> wurde seit Java 1.1 als deprecated deklariert)
MenuBar	Menüleiste (Container für Menüs) (Die Methode <code>countMenus()</code> wurde seit Java 1.1 als deprecated deklariert)
MenuComponent	Abstrakte Superklasse aller Menüelemente (Die Methoden <code>getPeer()</code> und <code>postEvent()</code> wurden seit Java 1.1 als deprecated deklariert)
MenuItem	Individuelle Menüoption (die Methoden <code>disable()</code> und <code>enable()</code> wurden seit Java 1.1 als deprecated deklariert)
MenuShortcut	Tastenkürzel für eine Menüoption (Java 1.1)
Panel	Container, der angezeigt wird
Point	Objekt, das einen Punkt darstellt (x- und y-Koordinaten)
Polygon	Objekt, das eine Gruppe von Punkten darstellt (Die Methoden <code>getBoundingBox()</code> und <code>inside()</code> wurden seit Java 1.1 als deprecated deklariert)
PopupMenu	Po-pup-Menü (Java 1.1)
PrintJob	Zu druckender Job (Java 1.1)
Rectangle	Objekt, das ein Rechteck darstellt (x- und y-Koordinaten für die obere Ecke plus Höhe und Breite) (Die Methoden <code>inside()</code> , <code>move()</code> , <code>reshape()</code> und <code>resize()</code> wurden seit Java 1.1 als deprecated deklariert)
RenderingHints	Implementierung der Map- und Cloneable-Schnittstellen, um Informationen über die Anzeige von zu zeichnenden Objekten zur Verfügung zu stellen
RenderingHints.Key	Innere Klasse von <code>RenderingHints</code> , die als Basisklasse für das Spezifizieren von Schlüsseln, die während des Rendering-Prozesses benutzt werden, dient
Scrollbar	Bildlaufleistenobjekt der Benutzeroberfläche (die folgenden Methoden wurden seit Java 1.1 als deprecated deklariert: <code>getLineIncrement()</code> , <code>getPageIncrement()</code> , <code>getVisible()</code> , <code>setLineIncrement()</code> und <code>setPageIncrement()</code> .)

ScrollPane	Container mit automatischem Rollen (die Methode layout() wurde seit Java 1.1 als deprecated deklariert)
SystemColor	Klasse, die die Farben der grafischen Benutzeroberfläche eines Systems enthält (Java 1.1)
TextArea	Mehrzeiliges scrollbares und editierbares Textfeld (die folgenden Methoden wurden seit Java 1.1 als deprecated deklariert: appendText(), insertText(), minimumSize(), preferredSize() und replaceText())
TextComponent	Superklasse aller editierbaren Textkomponenten
TextField	Editierbares Textfeld mit fester Größe (die Methoden minimumSize(), preferredSize(), setEchoCharacter() und resize() wurden seit Java 1.1 als deprecated deklariert)
TexturePaint	Verhaltensweisen zur Darstellung eines Texturfüllmusters in einer Form (Java 1.2)
Toolkit	Abstrakte Eigenschaft zum Binden der abstrakten AWT-Klassen in einer plattformspezifischen Toolkit-Implementierung
Window	Toplevel-Fenster und Superklasse der Klassen Frame und Dialog (die Methode postEvent() wurde seit Java 1.1 als deprecated deklariert)

Java.awt.color

Dieses Paket, das mit Java 1.2 eingeführt wurde, beschreibt unterschiedliche Systeme zur Beschreibung von Farben und ermöglicht die Konvertierung zwischen diesen Systemen.

Klassen

ColorSpace	Das spezielle Farbsystem eines Color-Objektes, eines Bildes oder eines Graphics-Objektes
ICC_ColorSpace	Ein Farbsystem, das auf der International Color Consortium (ICC) Profile Format Specification basiert
ICC_Profile	Eine Repräsentation von Farbprofildaten aus der ICC-Spezifikation
ICC_ProfileGray	Eine Untermenge von Profilen, wie z.B. solche, die bei monochromatischen Ein- und Ausgabegeräten verwendet werden
ICC_ProfileRGB	Eine Untermenge von Profilen, wie z.B. solche, die bei Drei-Komponenten-RGB-Ein- und -Ausgabegeräten verwendet werden

java.awt.datatransfer

Dieses Paket, das mit Java 1.1 eingeführt wurde, ist ein Teil des AWT-Paketes und enthält Schnittstellen und Methoden zur Kommunikation mit der Zwischenablage.

Schnittstellen

ClipboardOwner	Methoden für Objekte, die Daten für die Zwischenablage liefern
FlavorMap	Bildet MIME-Typen auf Java data flavors ab (Java 1.2)
Transferable	Methoden für Objekte, die Daten für eine Transferoperation liefern

Klassen

Clipboard	Die Zwischenablage selbst
DataFlavor	Die Objekte dieser Klasse stellen Datenformate dar
StringSelection	Transferagent für eine Zeichenkette
SystemFlavor-Map	Stellt eine Standardimplementierung der FlavorMap-Schnittstelle zur Verfügung (Java 1.2)

java.awt.dnd

Dieses Paket wurde mit Java 1.2 eingeführt und verarbeitet beide Seiten einer Drag&Drop-Aktion.

Schnittstellen

Autoscroll	Methoden, um automatisches Scrollen durch GUI-Elemente während Drag&Drop-Operationen zu ermöglichen (Java 1.2)
DragGesture-Listener	Schnittstelle, um das Ereignis DragGestureEvent zu behandeln (Java 1.2)
DragSourceListener	Methoden für die Quelle einer Drag&Drop-Operation, um damit verbundene Ereignisse zu behandeln
DropTargetListener	Methoden für das Ziel einer Drag&Drop-Operation, um damit verbundene Ereignisse zu behandeln
FlavorMap	Methoden, um MIME-Typen Strings zuzuordnen, die die nativen Plattformdatentypen identifizieren, wodurch der Drag&Drop-Transfer möglich wird (Java 1.1)

Klassen

DndConstants	Methoden, die zur Verarbeitung einer Drag&Drop-Operation verwendet werden
DragGestureEvent	Klasse, um ein Ereignis zu definieren, das signalisiert, daß ein Benutzer eine Drag&Drop-Operation begonnen hat (Java 1.2)
DragGesture-Recognizer	Klasse, um plattformabhängige Event Listener für Drag&Drop Operationen zu entwickeln (Java 1.2)

DragSource	Die Quelle einer Drag&Drop-Operation
DragSourceContext	Die Umgebung, die zur Ereignisweiterleitung verwendet wird und einen Transferable-Status bietet
DragSourceDragEvent	Ein Ereignis, das in der Quelle verwendet wird, um auf die Drag-Operation reagieren zu können
DragSourceDropEvent	Ein Ereignis, das in der Quelle verwendet wird, um auf die Drop-Operation reagieren zu können
DragSourceEvent	Superklasse der Klassen DragSourceDropEvent und DragSourceDragEvent
DropTarget	Ziel einer Drag&Drop-Operation
DropTarget.Drop-TargetAutoScroller	Innere Klasse von DropTarget, die Scroll-Operationen unterstützt (Java 1.2)
DropTargetContext	Umgebung des Ziels, die zur Ereignisweiterleitung verwendet wird und einen Transferable-Status bietet
DropTargetContext TransferableProxy	Verhaltensweisen zur Übertragung von Daten (Java 1.1)
DropTargetDragEvent	Ein Ereignis, das im Ziel verwendet wird, um auf die Drag-Operation reagieren zu können
DropTargetDropEvent	Ein Ereignis, das im Ziel verwendet wird, um auf die Drop-Operation reagieren zu können
DropTargetEvent	Superklasse der Klassen DropTargetDragEvent und DropTargetDropEvent
MouseDragGesture-Recognizer	Diese Klasse ist von DragGestureRecognizer abgeleitet und stellt Unterstützung für mausbasierte Drag&Drop Listener bereit (Java 1.2)

java.awt.event

Das Paket java.awt.event wurde mit Java 1.1 eingeführt und ist ein Teil des AWT- Paketes. Es enthält das Ereignismodell für Java 1.1 und höher.

Schnittstellen

ActionListener	Methoden zum Abfangen eines Aktionsereignisses
AdjustmentListener	Methoden zum Abfangen eines Justierungsereignisses
AWTEventListener	Methoden zum Abfangen eines AWT-Ereignisses (Java 1.2)
ComponentListener	Methoden zum Abfangen eines Komponentenereignisses
ContainerListener	Methoden zum Abfangen eines Containerereignisses
FocusListener	Methoden zum Abfangen eines Fokusereignisses
InputMethodListener	Methoden zum Abfangen eines Ereignisses
ItemListener	Methoden zum Abfangen eines Elementereignisses

KeyListener	Methoden zum Abfangen eines Tastaturereignisses
MouseListener	Methoden zum Abfangen eines Mausereignisses
MouseMotionListener	Methoden zum Abfangen eines Mausbewegungsereignisses
TextListener	Methoden zum Abfangen eines Textereignisses
WindowListener	Methoden zum Abfangen eines Fensterereignisses

Klassen

ActionEvent	Aktionsergebnis
AdjustmentEvent	Justierungsergebnis, das von einem Adjustable-Objekt erzeugt wurde
ComponentAdapter	Adapter, der Komponentenereignisse abfängt
ComponentEvent	Komponentenergebnis
ContainerAdapter	Adapter, der Container-Ereignisse abfängt
ContainerEvent	Containerergebnis
FocusAdapter	Adapter, der Fokusereignisse abfängt
FocusEvent	Fokusergebnis
InputEvent	Eingabeereignis
InputMethodEvent	Ereignis, das durch Änderungen an einem Text, der über eine Eingabe-Methode eingegeben wird, ausgelöst wird (Java 1.2)
InvocationEvent	Ereignis, das durch die Erzeugung eines Runnable-Objekts ausgelöst wird (Java 1.2)
ItemEvent	Elementergebnis, das von einem ItemSelectable-Objekt erzeugt wurde
KeyAdapter	Adapter, der Tastaturereignisse abfängt
KeyEvent	Tastaturergebnis
MouseAdapter	Adapter, der Mausereignisse abfängt
MouseEvent	Mausergebnis
MouseMotionAdapter	Adapter, der Mausbewegungsereignisse abfängt, z.B. Ziehen
PaintEvent	Zeichnenereignis auf Komponentenebene
TextEvent	Textergebnis, das von einer Komponente vom Typ TextComponent erzeugt wurde
WindowAdapter	Adapter, der Fensterereignisse abfängt
WindowEvent	Fensterergebnis

Java.awt.font

Dieses Paket, das mit Java 1.2 eingeführt wurde, unterstützt die Zusammenstellung, Anzeige von Schriften und den Umgang mit einzelnen Zeichen von Fonts.

Schnittstellen

MultipleMaster	Methoden zur Unterstützung von Typ-1-Multiple-Master-Fonts
OpenType	Methoden zur Unterstützung von TrueType- und OpenType-Fonts

Klassen

FontRendererContext	Stellt einen Container für die Information, die gebraucht wird, um Text richtig bemessen zu können, zur Verfügung (Java 1.2)
GlyphJustificationInfo	Ausrichtungseigenschaften eines Schriftzeichens
GlyphMetrics	Meßwerte eines Schriftzeichens
GlypSet	Eine grafische Repräsentation von Text (nur Java 1.1)
GlyphVector	Stellt Text als eine Folge von Integer Glyph Codes dar (Java 1.2)
GraphicAttribute	Wird benutzt, um eine Grafik zu identifizieren, die in Text eingebettet ist (Java 1.2)
ImageGraphicAttribute	Wird benutzt, um ein Bild zu identifizieren, das in Text eingebettet ist (Java 1.2)
LineBreakMeasurer	Organisiert Textzeilen entsprechend einer Packungsbreite (Java 1.2)
LineMetrics	Zugriff auf die zeilenorientierte Textmetrik (Java 1.2)
ShapeGraphicAttribute	Wird benutzt, um ein Shape-Objekt zu identifizieren, das in Text eingebettet ist (Java 1.2)
StyledString	Verhaltensweisen, die zur Anzeige von Text verwendet werden (nur Java 1.1)
StyledStringIterator	Eine Implementation des AttributedCharacterIterater-Protokolls für formatierten Text (nur Java 1.1)
TextAttribute	Attribute, die für das Textlayout benötigt werden
TextHitInfo	Eine Zeichenposition, um die Position des Cursors bzw. der Einfügemarke anzugeben
TextLayout	Unterstützung beim Layout von Text (Java 1.2)
TextLayout.Caret-Policy	Innere Klasse von TextLayout, die angibt, wie das Caret mit einem TextLayout-Objekt benutzt werden soll (Java 1.2)
TextLine.TextLine-Metrics	Stellt Basis-Metriken für das Arbeiten mit Text zur Verfügung (Java 1.2)
TransformAttribute	Ermöglicht, Transformationen als Attribute zu benutzen (Java 1.2)

java.awt.geom

Dieses Paket, das mit Java 1.2 eingeführt wurde, unterstützt 2D-Geometrie.

Schnittstellen

PathIterator	Methoden, die an der Pfaditeration beteiligt sind

Klassen

AffineTransformation	Affine 2D-Transformationen (die Methoden prepend() und append() wurden seit Java 1.2 als deprecated deklariert)
Arc2D	Bögen, die über das umgebende Rechteck, den Anfangswinkel, den Bogenbereich und die Schließart definiert werden
Arc2D.Double	Bögen, die mit doppelter Fließkomma-Genauigkeit definiert werden (Java 1.2)
Arc2D.Float	Bögen, die mit Fließkomma-Genauigkeit definiert werden
Area	Eine geometrische Fläche
CubicCurve2D	Segmente von kubischen Parameterkurven im (x,y)-Koordinatenraum
CubicCurve2D.Double	Segmente von kubischen Parameterkurven im (x,y)-Koordinatenraum mit double Koordinaten (Java 1.2)
CubicCurve2D.Float	Segmente von kubischen Parameterkurven im (x,y)-Koordinatenraum mit float Koordinaten
Dimension2D	Die Repräsentation der Dimensionen Breite und Höhe
Ellipse2D	Ellipsen, die über das umgebende Rechteck definiert werden
Ellipse2D.Double	Ellipsen, die mit doppelter Fließkomma-Genauigkeit definiert werden (Java 1.2)
Ellipse2D.Float	Ellipsen, die mit Fließkomma-Genauigkeit definiert werden
FlatteningPathIterator	Ebenenprojektion eines anderen PathIterator-Objekts
GeneralPath	Geometrische Pfade, die aus geraden Linien sowie quadratischen und kubischen Kurven konstruiert sind
GeneralPathIterator	Ein Iterator für GeneralPath-Objekte (nur Java 1.1)
Line2D	Liniensegmente im (x,y)-Koordinatenraum
Line2D.Double	Liniensegmente im (x,y)-Koordinatenraum mit double-Koordinaten (Java 1.2)
Line2D.Float	Liniensegmente im (x,y)-Koordinatenraum mit float-Koordinaten
Point2D	Punkte im (x,y)-Koordinatenraum
Point2D.Double	Punkte im (x,y)-Koordinatenraum mit double-Koordinaten
Point2D.Float	Punkte im (x,y)-Koordinatenraum mit float-Koordinaten (Java 1.2)

QuadCurve2D	Segmente quadratischer Kurven im (x,y)-Koordinatenraum
QuadCurve2D.Double	Segmente quadratischer Kurven im (x,y)-Koordinatenraum mit double-Koordinaten (Java 1.2)
QuadCurve2D.Float	Segmente quadratischer Kurven im (x,y)-Koordinatenraum mit float-Koordinaten
Rectangle2D	Rechtecke, die über (x,y)-Koordinaten und Dimensionen definiert werden
Rectangle2D.Double	Rechtecke, die mit double-Koordinaten definiert werden
Rectangle2D.Float	Rechtecke, die mit float-Koordinaten definiert werden (Java 1.2)
RectangularShape	Figuren, die in ein Rechteck einbeschrieben werden können
RoundRectangle2D	2D-Rechtecke, die Bögen als Ecken haben
RoundRectangle2D. Double	2D-Rechtecke, die Bögen als Ecken haben - definiert in double-Koordinaten (Java 1.2)
RoundRectangle2D. Float	2D-Rechtecke, die Bögen als Ecken haben - definiert in float-Koordinaten

java.awt.im

Dieses Paket wurde mit Java 1.2 eingeführt und bietet Eingabemethoden, die zur Eingabe von Tausenden unterschiedlicher Tasten-Sequenzen, die sich aus mehreren Tastenanschlägen zusammensetzen, dienen. Eingabemethoden ermöglichen die Eingabe von chinesischen, japanischen und koreanischen Texten in Textkomponenten.

Schnittstellen

InputMethodRequest	Methoden, die es Textkomponenten ermöglichen, mit Eingabemethoden zusammenzuarbeiten

Klassen

InputContext	Verhaltensweisen zur Steuerung der Kommunikation zwischen Eingabemethoden und Textkomponenten
InputMethodHighlight	Hervorhebungsstile der Eingabemethoden für den eingegebenen Text
InputSubset	Unicode-Unterstützung für die Eingabemethoden (Java 1.2)

java.awt.image

Das Paket java.awt.image ist ein Teil des AWT-Paketes und enthält Schnittstellen und Klassen zur Verwaltung von Bitmap-Bildern.

Schnittstellen

BufferedImageOp	Methoden, um Operationen für BufferedImage-Objekte zu beschreiben (Java 1.2)
ImageConsumer	Methoden zum Empfang eines von einem ImageProducer erzeugten Bildes
ImageObserver	Methoden zur Verfolgung des Ladens und des Aufbaus eines Bildes
ImageProducer	Methoden zum Produzieren von Bilddaten, die von einem ImageConsumer empfangen werden
RasterOp	Methoden, um Operation für Raster-Objekte zu beschreiben (Java 1.2)
RenderedImage	Methoden für Objekte, die Raster erzeugen oder beinhalten (Java 1.2)
TileObserver	Methoden zur Überwachung, wann Bereiche eines WritableRenderedImage-Objekts verändert werden (Java 1.2)
WritableRenderedImage	Methoden für Objekte, die veränderbare bzw. überschreibbare Bilder produzieren oder beinhalten (Java 1.2)

Klassen

AffineTransformOp	Affine Transformationen, um Bilder oder Raster linear aufeinander abzubilden (Java 1.2)
AreaAveragingScaleFilter	Ein Filter, der ein Bild unter Verwendung von Flächenmittelwerten skaliert (Java 1.2).
BandCombineOp	Willkürliche Kombination von Rasterbändern unter Verwendung von vorgegebenen Matrizen (Java 1.2)
BandedSampleModel	Verhaltensweisen, um Bilderdaten zu speichern. Die einzelnen Datenbänder werden in verschiedenen Bänken eines DataBuffer-Objekts gespeichert (Java 1.2)
BufferedImage	Bilder mit einem Datenpuffer, auf den zugegriffen werden kann (Java 1.2; die Methode <code>getGraphics()</code> wurde seit Java 1.2 als deprecated deklariert)
BufferedImageFilter	Filter, die gepufferte Bilddaten von ImageProducer-Objekten entgegennehmen und veränderte Versionen davon an ImageConsumer-Objekte weitergeben (Java 1.2)
ByteLookupTable	Tabellen zum Ermitteln von Informationen aus Bildbereichskanälen oder Bildkomponentenstrukturen, wie z.B. RGB (Java 1.2)
ColorConvertOp	Pixelweise Farbkonvertierungen von Quell-/Bildern (Java 1.2)
ColorModel	Abstrakte Klasse zur Verwaltung von Farbinformationen für Bilder
ComponentColor-Model	Verhaltensweisen, um Farbinformationen für Bilder zu verwalten, die willkürliche ColorSpace-Objekte verwenden (Java 1.2)

ComponentSampleModel	Verhaltensweisen, um Bilder zu sampeln, die Daten in getrennten Datenarrayelementen derselben <code>DataBuffer</code> -Bank speichern (Java 1.2)
ConvolveOp	Berechnet aus einem Quellbild ein Zielbild unter Verwendung einer Matrix, die die Art der Berechnung beschreibt (Java 1.2)
CropImageFilter	Filter zum Beschneiden von Bildern auf eine bestimmte Größe
DataBuffer	Kapselung eines oder mehrerer Datenarrays in Bänke (Java 1.2)
DataBufferByte	Datenpuffer, die als <code>byte</code> -Werte gespeichert werden (Java 1.2)
DataBufferInt	Datenpuffer, die als <code>int</code> -Werte gespeichert werden (Java 1.2)
DataBufferShort	Datenpuffer, die als <code>short</code> -Werte gespeichert werden (Java 1.2)
DataBufferUShort	Datenpuffer, die als <code>unsigned short</code> -Werte gespeichert werden (Java 1.2)
DirectColorModel	Spezifisches Farbmodell zum Verwalten und Umsetzen von Pixelfarbwerten
FilteredImageSource	Ein <code>ImageProducer</code> , der aus einem Bild und einem <code>ImageFilter</code> -Objekt ein Bild für einen <code>ImageConsumer</code> erzeugt
ImageFilter	Filter, der Bilddaten aus einem <code>ImageProducer</code> entnimmt, sie auf bestimmte Weise ändert und an einen <code>ImageConsumer</code> weitergibt
IndexColorModel	Spezifisches Farbmodell zur Verwaltung und Umsetzung von Farbwerten in einer Palette mit feststehenden Farben
Kernel	Matrizen, die beschreiben, wie ein Pixel und dessen umgebende Pixel den Wert des Pixels bei einer Filteroperation beeinflussen (Java 1.2)
LookupOp	Operationen, bei denen Daten aus einem <code>LookupTable</code> -Objekt ermittelt werden
LookupTable	Tabellen, die Daten für Bildbereichskanäle oder Bildkomponentenstrukturen enthalten (Java 1.2)
MemoryImageSource	Ein <code>ImageProducer</code> , der ein Bild aus dem Speicher erhält (zum manuellen Erstellen eines Bildes)
MultiPixelPackedSampleModel	Verhaltensweisen zum Sampeln eines Bildes, bei dem mehrere Pixel mit je einem Sample verarbeitet werden (Java 1.2)
PackedColorModel	Ein Farbmodell, bei dem die Farbwerte der einzelnen Pixel direkt in den Werten der Pixel gespeichert werden (Java 1.2)
PixelGrabber	Ein <code>ImageConsumer</code> , der eine Untergruppe der Pixel in einem Bild erhält
PixelInterleaved-SampleModel	Methoden, um Bilddaten in der Form von interleaved Pixeln speichern zu können (Java 1.2)
Raster	Eine Raster-Repräsentation von Bilddaten (Java 1.2)

<code>ReplicateScaleFilter</code>	Ein Filter, der ein Bild skaliert (Java 1.1)
<code>RescaleOp</code>	Pixelweise Skalierung von Bildern oder Rastern (Java 1.2)
<code>RGBImageFilter</code>	Filter, der RGB-Bilddaten aus einem <code>ImageProducer</code> entnimmt, sie auf bestimmte Weise ändert und an einen <code>ImageConsumer</code> weitergibt
<code>SampleModel</code>	Basisklasse, um Algorithmen zum Bilddaten-Sampling zu entwickeln (Java 1.2)
<code>ShortLookupTable</code>	Tabellen, die <code>short</code> -Daten für Bildbereichskanäle oder Bildkomponentenstrukturen enthalten (Java 1.2)
<code>SinglePixelPackedSampleModel</code>	Verhaltensweisen zum Samplen eines Bildes, dessen Pixel sich nicht über mehrere Datenelemente erstrecken (Java 1.2)
<code>WritableRaster</code>	Eine überschreibbare Repräsentation von <code>image</code> -Daten (Java 1.2)

java.awt.image.renderable

Dieses Paket wurde mit Java 1.2 eingeführt und bietet Klassen und Schnittstellen für das Bild-Rendering.

Schnittstellen

<code>ContextualRendered-ImageFactory</code>	Methoden, die Rendering-unabhängige Aufgaben unterstützen
<code>RenderableImage</code>	Methoden, um Bildoperationen auszuführen, die unabhängig von jeglichem speziellen Bild-Rendering sind
<code>RenderedImageFactory</code>	Diese Schnittstelle definiert die <code>create()</code> -Methode, die von Klassen, die verschiedene Bild-Render-Methoden implementieren, benutzt wird

Klassen

<code>ParameterBlock</code>	Diese Klasse beinhaltet einen allgemeinen Satz von Parametern für Objekte vom Typ <code>RenderableImageOp</code>
<code>RenderContext</code>	Diese Klasse spezifiziert Kontextinformationen für das Rendern eines <code>RenderableImage</code> -Objekts
<code>RenderableImageOp</code>	Diese Klasse unterstützt das kontextspezifische Bild-Rendering
<code>RenderableImage-Producer</code>	Diese Klasse unterstützt die asynchrone Erstellung eines <code>RenderableImage</code> -Objekts

java.awt.peer (nur Java 1.1)

Das Paket `java.awt.peer` ist ein Teil des AWT-Paketes und enthält die (verborgenen) plattformspezifischen AWT-Klassen (z.B. für Motif, Macintosh, Windows 95) mit plattformunabhängigen Schnittstellen. Beim Aufruf dieser Schnittstellen muß das Fenstersystem der jeweiligen Plattform bekannt sein.

Jede Klasse des AWT, die von `Component` oder `MenuComponent` abgeleitet wurde, hat eine entsprechende `peer`-Klasse. Diese Klassen werden mit dem Namen aus `Component` mit dem Zusatz `-Peer` bezeichnet (z.B. `ButtonPeer`, `DialogPeer` und `WindowPeer`). Sie werden hier nicht einzeln aufgeführt, da sie alle die gleichen Eigenschaften haben.

java.awt.print

Dieses Paket wurde mit Java 1.2 eingeführt und beinhaltet Schnittstellen und Klassen für das Drucken.

Schnittstellen

<code>PrinterGraphics</code>	Methoden, um auf ein <code>PrinterJob</code> -Objekt zuzugreifen
<code>Pageable</code>	Methoden, die einen Satz von Seiten, die gedruckt werden sollen, repräsentieren
<code>Printable</code>	Methoden, die in Verbindung mit <code>PageFormat</code> -Objekten zum Druck dienen

Klassen

<code>Book</code>	Eine Liste von Seiten zum Druck
<code>PageFormat</code>	Die Größe und Ausrichtung von Seiten für den Druck
<code>Paper</code>	Physische Eigenschaften des Druckpapiers
<code>PrinterJob</code>	Ein Druckjob

java.beans

Das Paket `java.beans`, das mit Java 1.1 eingeführt wurde, enthält die Klassen und Schnittstellen, die die JavaBeans-Technologie ermöglichen.

Schnittstellen

<code>AppletInitializer</code>	Methoden, die die ordnungsgemäße Initialisierung von Bean-Applets ermöglichen (Java 1.2)
<code>BeanInfo</code>	Methoden zur Informationsbeschaffung, die speziell von einem Bean geliefert werden

Customizer	Methoden zur Definition des Overheads für einen kompletten visuellen Editor für ein Bean
DesignMode	Diese Schnittstelle wird benutzt, um anzuzeigen, daß sich ein Bean im Design-Modus (als Gegenteil zum Ausführungs-Modus) befindet (Java 1.2)
PropertyChangeListener	Methode, die bei Änderung einer gebundenen Eigenschaft aufgerufen wird
PropertyEditor	Methoden zur Unterstützung von grafischen Benutzeroberflächen, die dem Benutzer das Editieren eines Wertes eines bestimmten Typs ermöglichen
VetoableChangeListener	Methoden, die bei Änderung einer Constraint-Eigenschaft aufgerufen werden
Visibility	Methoden zur Bestimmung, ob ein Bean eine grafische Benutzeroberfläche fordert und ob eine grafische Benutzeroberfläche für das Bean zur Verfügung steht

Klassen

BeanDescriptor	Liefert globale Informationen über ein Bean
Beans	Liefert einige Mehrzweckkontrollmethoden für Beans
EventSetDescriptor	Stellt eine Gruppe von Ereignissen dar, die ein Bean erzeugen kann
FeatureDescriptor	Dient als gemeinsame Basisklasse für die Klassen EventSetDescriptor, MethodDescriptor und PropertyDescriptor
IndexedPropertyDescriptor	Liefert Methoden für den Zugriff auf den Typ einer Index-Eigenschaft zusammen mit dessen Zugriffsmethoden
Introspector	Liefert den zur Analyse eines Beans erforderlichen Overhead und bestimmt dessen öffentliche Eigenschaften, Methoden und Ereignisse
MethodDescriptor	Liefert Methoden für den Zugriff auf Informationen wie z.B. die Parameter einer Methode
ParameterDescriptor	Ermöglicht Bean-Implementoren die Bereitstellung zusätzlicher Informationen zu den einzelnen Parametern
PropertyChangeEvent	Ereignis, das bei einer Änderung einer gebundenen Eigenschaft auftritt
PropertyChangeSupport	Eine Hilfsklasse zur Verwaltung der Listeners von gebundenen Eigenschaften
PropertyDescriptor	Liefert Methoden für den Zugriff auf den Typ einer Eigenschaft zusammen mit den Zugriffsmethoden
PropertyEditorManager	Liefert eine Möglichkeit zur Registrierung von Eigenschaftstypen, so daß ihre Editoren leicht zu finden sind
PropertyEditorSupport	Eine Hilfsklasse, die die PropertyEditor-Schnittstelle implementiert, die die Erstellung individueller Eigenschaftseditoren etwas erleichtern soll

SimpleBeanInfo	Eine Supportklasse, die es dem Bean-Entwickler erleichtern soll, explizite Informationen über ein Bean zu liefern
VetoableChangeSupport	Eine Hilfsklasse zur Verwaltung von Listeners für die gebundene und Constrained-Eigenschaften

java.beans.beancontext

Dies ist ein Paket, das mit Java 1.2 eingeführt wurde, um allgemeine Dienstmechanismen für Beans zu bieten.

Schnittstellen

BeanContext	Methoden, um Informationen über den Status der Umgebung eines Beans zur Verfügung zu stellen
BeanContextChild	Methoden, um die Laufzeitumgebung eines Beans festzulegen
BeanContextChild-ComponentProxy	Schnittstelle für den Zugriff auf die AWT-Komponente, die mit dem BeanContextChildren-Objekt verknüpft ist
BeanContextContainer-Proxy	Diese Schnittstelle ermöglicht den Zugriff auf den AWT-Container, der mit einem BeanContext-Objekt verknüpft ist
BeanContextMembershipListener	Methoden, um auf BeanContext-Ereignisse zu reagieren
BeanContextProxy	Diese Schnittstelle wird von Beans implementiert, die den Context anderer Beans benutzen
BeanContextService-Provider	Methoden, um Services einem BeanContext zur Verfügung zu stellen
BeanContextService-ProviderBeanInfo	Genaue Informationen über die Services einer Schnittstelle
BeanContextService-RevokedListener	Behandlung von Ereignissen, die mit dem Widerruf eines Services gegenüber einem BeanContext verbunden sind
BeanContextServices	Methoden, die es einem BeanContext-Objekt erlauben, Services den in ihm enthaltenen BeanContextChild-Objekten zur Verfügung zu stellen
BeanContextServices-Listener	Methoden, um Ereignisse zu behandeln, die damit, daß ein Service für einen BeanContext verfügbar wird, in Verbindung stehen

Klassen

BeanContextChild-Support	Basis-Implementierung der BeanContextChild-Schnittstelle

BeanContextEvent	Ereignisse, die erzeugt werden, wenn sich der Status des BeanContext verändert
BeanContextMembershipEvent	Ereignisse, die an alle Mitglieder gesendet werden, wenn sich der Status ändert
BeanContextService-AvailableEvent	Zeigt an, daß ein Service einem BeanContext zur Verfügung gestellt wurde
BeanContextService-RevokedEvent	Zeigt an, daß ein Service einem BeanContext nicht weiter zur Verfügung steht
BeanContextServices-Support	Stellt eine Basis-Implementierung der BeanContextServices-Schnittstelle zur Verfügung
BeanContextServices-Support.BCSSChild	Innere Klasse von BeanContextServicesSupport, die von BeanContextSupport geerbt wird.
BeanContextSupport	Implementierung der Schnittstelle BeanContext
BeanContextSupport. BCSIterator	Innere Klasse von BeanContextSupport, die als Iterator innerhalb ihrer Elternklasse dient

java.io

Das Paket java.io enthält Ein- und Ausgabeklassen und Schnittstellen für Streams und Dateien.

Schnittstellen

DataInput	Methoden zum Lesen von maschinenunabhängigen Eingabestreams
DataOutput	Methoden zum Schreiben von maschinenunabhängigen Ausgabestreams
Externalizable	Methoden zum Lesen/Schreiben des Inhalts eines Objektes mit einem Stream (Java 1.1)
FileFilter	Methoden zum Filtern von Pfadnamen
FilenameFilter	Methoden zum Filtern von Dateinamen
ObjectInput	Methoden zum Lesen von Objekten (Java 1.1)
ObjectInputValidation	Methoden zur Prüfung eines Objektes (Java 1.1)
ObjectOutput	Methoden zum Schreiben von Objekten (Java 1.1)
ObjectStreamConstants	Konstanten, die bei der objektbasierten Ein- und Ausgabe benutzt werden (Java 1.2)
Serializable	Marke zur Anzeige, daß diese Klasse serialisiert werden kann (Java 1.1)

Klassen

BufferedInputStream	Gepufferter Eingabestream
BufferedOutputStream	Gepufferter Ausgabestream

BufferedReader	Gepufferter Reader (Java 1.1)
BufferedWriter	Gepufferter Writer (Java 1.1)
ByteArrayInputStream	Eingabestream aus einem Byte-Array
ByteArrayOutputStream	Ausgabestream an ein Byte-Array (die Methode toString() wurde seit Java 1.1 als deprecated deklariert)
CharArrayReader	Reader aus einem Zeichen-Array (Java 1.1)
CharArrayWriter	Writer an ein Zeichen-Array (Java 1.1)
DataInputStream	Ermöglicht das maschinenunabhängige Lesen von Java-Primitivtypen (int, char, boolean usw.) aus einem Stream (die Methode readLine() wurde seit Java 1.1 als deprecated deklariert)
DataOutputStream	Ermöglicht das maschinenunabhängige Schreiben von Java-Primitivtypen (int, char, boolean usw.) in einen Stream
File	Stellt eine Datei des Host-Dateisystems dar
FileDescriptor	Hält den Unix-ähnlichen Bezeichner einer Datei oder eines Sockets
FileInputStream	Eingabestream von einer Datei, der mit einem Dateinamen oder Bezeichner erstellt wird
FileOutputStream	Ausgabestream an eine Datei, der mit einem Dateinamen oder Bezeichner erstellt wird
FilePermission	Kontrolliert die Zugriffsrechte für Dateien
FileReader	Ein Reader für eine Datei, der einen Dateinamen oder Bezeichner nutzt (Java 1.1)
FileWriter	Ein Writer in eine Datei, der einen Dateinamen oder Bezeichner nutzt (Java 1.1)
FilterInputStream	Abstrakte Klasse, die als Filter für die Eingabestreams (und zum Hinzufügen von Streamfunktionen, z.B. Puffern) dient
FilterOutputStream	Abstrakte Klasse, die als Filter für die Ausgabestreams (und zum Hinzufügen von Streamfunktionen, z.B. Puffern) dient
FilterReader	Klasse, die als Filter für die Reader (und zum Hinzufügen von Streamfunktionen, z.B. Puffern) dient (Java 1.1)
FilterWriter	Klasse, die als Filter für die Writer (und zum Hinzufügen von Streamfunktionen, z.B. Puffern) dient (Java 1.1)
InputStream	Abstrakte Klasse, die einen Byte-Eingabestream darstellt (die »Mutter« aller Eingabestreams dieses Pakets)
InputStreamReader	Ein Reader, der eine Brücke zwischen Byte-Streams und Zeichen-Streams schafft (Java 1.1)
LineNumberInputStream	Ein Eingabestream zum Erzeugen von Zeilennummern (wurde seit Java 1.2 als deprecated deklariert)
LineNumberReader	Ein gepufferter Zeicheneingabe-Stream, der die Zeilennummern verfolgt (Java 1.1)

ObjectInputStream	Eine Klasse zur Entserialisierung von Daten und Objekten (Java 1.1; die Methode readLine() wurde seit Java 1.2 als deprecated deklariert)
ObjectInputStream.GetField	Unterstützung für das Auslesen von einzelnen Objekt-Feldern (Java 1.2)
ObjectOutputStream	Eine Klasse zur Serialisierung von Daten und Objekten (Java 1.1)
ObjectOutputStream.PutField	Unterstützung für den Zugriff auf einzelne Objekt-Felder (Java 1.2)
ObjectStreamClass	Ein Bezeichner für serialisierbare Klassen (Java 1.1)
ObjectStreamField	Beschreibt ein Feld einer serialisierten Klasse
OutputStream	Abstrakte Klasse, die einen Byte-Ausgabestream darstellt (die »Mutter« aller Ausgabestreams dieses Pakets)
OutputStreamWriter	Eine Brücke zwischen Byte- und Zeichenstreams (Java 1.1)
PipedInputStream	Ein Pipe-Eingabestream, der mit einem entsprechenden PipedOutputStream verbunden sein muß
PipedOutputStream	Eine Pipe-Ausgabestream, der mit einem entsprechenden PipedInputStream verbunden sein muß (zusammen bieten diese eine sichere Kommunikation zwischen Threads)
PipedReader	Ein Pipe-Reader, der mit einem entsprechenden PipedWriter verbunden sein muß (Java 1.1)
PipedWriter	Ein Pipe-Writer, der mit einem entsprechenden PipedReader verbunden sein muß (Java 1.1)
PrintStream	Ein Ausgabestream zum Drucken - Verwendung in System.out.println(...) (die Konstruktoren PrintStream() wurden seit Java 1.1 als deprecated deklariert)
PrintWriter	Ein Writer zum Drucken (Java 1.1)
PushbackInputStream	Ein Eingabestream mit einem Rückstellpuffer
PushbackReader	Ein Reader mit einem Rückstellpuffer (Java 1.1)
RandomAccessFile	Ermöglicht den wahlfreien Zugriff auf eine Datei und wird aus Dateinamen, Bezeichnern oder Objekten erstellt
Reader	Abstrakte Klasse, die einen Zeichen-Eingabestream darstellt (die »Mutter« aller Reader dieses Pakets) (Java 1.1)
SequenceInputStream	Konvertiert eine Serie von Eingabestreams in einen einzelnen Eingabestream
SerializablePermission	Verhaltensweisen, um die Zugriffskontrolle bei der Serialisierung zu handhaben (Java 1.2)
StreamTokenizer	Konvertiert einen Eingabestrom in eine Serie von einzelnen Token (der Konstruktor StreamTokenizer() wurde seit Java 1.1 als deprecated deklariert)
StringBufferInputStream	Ein Eingabestream von einem String-Objekt (wurde seit Java 1.2 als deprecated deklariert)
StringReader	Ein Reader, der aus einem String-Objekt liest (Java 1.1)

StringWriter	Ein Writer, der in ein String-Objekt schreibt (Java 1.1)
Writer	Abstrakte Klasse, die einen Zeichen-Ausgabestream darstellt (die »Mutter« aller Writer dieses Pakets) (Java 1.1)

java.lang

Im Paket java.lang befinden sich die Klassen und Schnittstellen, die den Kern der Sprache Java bilden.

Schnittstellen

Cloneable	Schnittstelle zur Angabe, daß ein Objekt kopiert oder geklont werden kann
Comparable	Methoden, um den Objekten einer Klasse eine Ordnung aufzuerlegen (Java 1.2)
Runnable	Methode für Klassen, die als Threads laufen

Klassen

Boolean	Objekt-Wrapper für boolean-Werte
Byte	Objekt-Wrapper für byte-Werte (Java 1.1)
Character	Objekt-Wrapper für char-Werte (die Methoden isJavaLetter(), isJavaLetterOrDigit() und isSpace() wurden seit Java 1.1 als deprecated deklariert)
Character.Subset	Innere Klasse von Character, die Unicode-Konstanten definiert (Java 1.2)
Character.UnicodeBlock	Innere Klasse von Character für die Unicode-Unterstützung (Java 1.2)
Class	Darstellung von Klassen zur Laufzeit
ClassLoader	Abstrakte Eigenschaft zum Laden von Klassen (die Methode defineClass() wurde seit Java 1.1 als deprecated deklariert)
Compiler	Systemklasse, die Zugang zum Java-Compiler bietet
Double	Objekt-Wrapper für double-Werte
Float	Objekt-Wrapper für float-Werte
InheritableThreadLocal	Unterstützung für die Vererbung von Thread-Werten (Java 1.2)
Integer	Objekt-Wrapper für int-Werte
Long	Objekt-Wrapper für long-Werte
Math	Utility-Klasse für mathematische Operationen
Number	Abstrakte Superklasse aller Zahlenklassen (Integer, Float usw.)
Object	Allgemeine Objektklasse an oberster Stelle der Vererbungshierarchie
Package	Versionsinformationen über die Implementation und Spezifikation des Java-Paketes (Java 1.2)

Process	Abstrakte Eigenschaften für Prozesse der System-Klasse
Runtime	Zugriff auf die Java-Laufzeitumgebung (die Methoden <code>getLocalizedInputStream()</code> und <code>getLocalizedOutputStream()</code> wurden seit Java 1.1 als deprecated deklariert)
RuntimePermission	Verhaltensweisen, um Zugriffskontrolle zur Laufzeit zu ermöglichen (Java 1.2)
SecurityManager	Abstrakte Eigenschaften zur Implementierung von Sicherheitsmaßnahmen (mit Java 1.2 wurden die Variable <code>inCheck</code> und einige Methoden als deprecated deklariert: <code>getInCheck()</code> , <code>classDepth()</code> , <code>classLoaderDepth()</code> , <code>inClass()</code> und <code>inClassLoader()</code>)
Short	Objekt-Wrapper für short-Werte (Java 1.1)
String	Zeichenkette (Zwei <code>String()</code> -Konstruktoren und eine <code>getBytes()</code> wurden seit Java 1.1 als deprecated deklariert)
StringBuffer	Editierbare Zeichenkette
System	Zugriff auf Eigenschaften der Java-Systemebene, die in plattformunabhängiger Form angegeben werden (die Methode <code>getEnv()</code> wurde seit Java 1.1 als deprecated deklariert)
Thread	Methoden zur Verwaltung von Threads und Klassen, die in Threads laufen (die Methoden <code>stop()</code> , <code>suspend()</code> und <code>resume()</code> wurden seit Java 1.2 als deprecated deklariert)
ThreadDeath	Ein Objekt dieser Klasse wird ausgeworfen, wenn ein Thread asynchron beendet wird (nur Java 1.1)
ThreadGroup	Eine Gruppe von Threads (die Methoden <code>stop()</code> , <code>suspend()</code> und <code>resume()</code> wurden seit Java 1.2 als deprecated deklariert)
ThreadLocal	Verhaltensweisen, um Threads voneinander unabhängige individuelle Variablen zur Verfügung zu stellen
Throwable	Allgemeine Basisklasse für Ausnahmen; alle ausgeworfenen Objekte müssen vom Typ <code>Throwable</code> oder einer Subklasse davon sein
Void	Objekt-Wrapper für void-Typen (Java 1.1)

java.lang.ref

Dieses Paket, das mit Java 1.2 eingeführt wurde, ermöglicht es Objektreferenzen, gekapselt und untersucht zu werden wie andere Objekte auch.

Klassen

PhantomReference	Phantom Referenzobjekte
Reference	Abstraktion von Referenzobjekten (kann nicht instanziiert werden)
ReferenceQueue	Die Warteschlange, in die Referenzobjekte von dem Garbage Collector eingereiht werden

SoftReference	Gecachte schwache Referenzobjekte
WeakReference	Schwache Referenzobjekte

java.lang.reflect

Dieses Paket wurde mit Java 1.1 eingeführt, um die Reflection-Funktionalität zu unterstützen. Reflection ist eine Methode, um Informationen über eine geladene Klasse herauszufinden (wie z.B. deren Attribute und Verhaltensweisen).

Schnittstellen

Member	Methoden, um Informationen über ein Mitglied zu ermitteln

Klassen

AccessibleObject	Basisklasse für Field-, Method- und Constructor-Objekte (Java 1.2)
Array	Methoden, um Arrays dynamisch zu erstellen und dynamisch auf diese zuzugreifen
Constructor	Methoden, um Informationen über Konstruktoren zu ermitteln und auf diese zuzugreifen
Field	Methoden, um Informationen über Variablen zu ermitteln und auf diese zuzugreifen
Method	Methoden, um Informationen über Methoden zu ermitteln und auf diese zuzugreifen
Modifier	Decoder für Modifier
ReflectPermission	Verhaltensweisen, um Zugriffskontrolle für Reflection-Operationen zur Verfügung zu stellen (Java 1.2)

java.math

Das Paket java.math enthält zwei Klassen, die Zahlen extremer Größe aufnehmen können.

Klassen

BigDecimal	Sehr große Fließkommazahl
BigInteger	Sehr große Ganzzahl

java.net

Das Paket java.net enthält Klassen und Schnittstellen zur Durchführung von Netzoperationen, z.B. für Sockets und URLs.

Schnittstellen

ContentHandlerFactory	Methoden zum Erstellen von ContentHandler-Objekten
FileNameMap	Methoden zum Zuordnen von Dateinamen und MIME-Typen (Java 1.1)
SocketImplFactory	Methoden zum Erstellen von Socket-Implementierungen (Instanz der Klasse SocketImpl)
SocketOptions	Konstanten, um eine maßgeschneiderte Socketkonfiguration zu erstellen (Java 1.2)
URLStreamHandlerFactory	Methoden zum Erstellen von URLStreamHandler-Objekten

Klassen

Authenticator	Ein Objekt, das eine Netzwerk-Authentifizierung bekommen kann (Java 1.2)
ContentHandler	Abstrakte Eigenschaften zum Lesen von Daten in einer URL-Verbindung und Zusammenstellen des entsprechenden lokalen Objekts auf der Grundlage von MIME-Typen
DatagramPacket	Datagrammpaket (UDP)
DatagramSocket	Datagramm-Socket
DatagramSocketImpl	Abstrakte Basisklasse für Datagramm und Multicast-Sockets (Java 1.1)
URLConnection	Verbindung, die das HTTP-Protokoll verarbeiten kann (Java 1.1)
InetAddress	Eine Objektdarstellung eines Internet-Hosts (Hostname, IP-Adresse)
JarURLConnection	Eine URL-Verbindung zu einer .JAR-Datei oder einem Eintrag in einer .JAR-Datei (Java 1.2)
MulticastSocket	Serverseitiger Socket mit Unterstützung für die Übertragung von Daten an mehrere Client-Sockets (Java 1.1)
NetPermission	Verhaltensweisen, um Zugriffskontrolle für Netzwerke zu bieten (Java 1.2)
PasswordAuthentication	Unterstützung für die Authentifizierung im Netzwerk per Paßwort (Java 1.2)
ServerSocket	Serverseitiger Socket
Socket	Socket (zwei Socket()-Konstruktoren wurden seit Java 1.1 als deprecated deklariert)
SocketImpl	Abstrakte Klasse zur spezifischen Socketimplementierung

SocketPermission	Verhaltensweisen, um Zugriffskontrolle für ein Netzwerk über Sockets zu bieten (Java 1.2)
URL	Objektdarstellung eines URL
URLClassLoader	Der Class- und Jar-Archive-Loader eines URL-Suchpfades (Java 1.2)
URLConnection	Abstrakte Eigenschaften für einen Socket, der verschiedene Web-basierte Protokolle handhaben kann (http, ftp usw.)
URLDecoder	Dekodiert die von URLEncoder erstellten Zeichenketten (Java 1.2)
URLEncoder	Konvertiert Zeichenketten in das x-www-form-urlencoded-Format
URLStreamHandler	Abstrakte Klasse zur Verwaltung von Streams zu Objekten, die über URLs angegeben werden

java.rmi

Das Paket java.rmi enthält Klassen und Schnittstellen, die dem Programmierer die Erstellung verteilter Java-zu-Java-Applikationen ermöglicht, bei denen die Methoden entfernter Java-Objekte von anderen Maschinen, evtl. verschiedenen Hosts, aufgerufen werden können.

Schnittstellen

Remote	Methoden zur Identifizierung aller entfernten Objekte

Klassen

MarshaledObject	Ein Byte-Stream mit einer serialisierten Repräsentation eines Objektes, das seinem Konstruktor zur Verfügung gestellt wird (Java 1.2)
Naming	Methoden zur Ermittlung von Referenzen auf Remote-Objekte auf der Basis der Uniform-Resource-Locator-(URL-)Syntax
RMISecurityManager	Methoden zur Definition der RMI-Stub-Sicherheitsmethode für Applikationen (nicht für Applets)

java.rmi.activation

Dieses mit Java 1.2 eingeführte Objekt unterstützt persistente Referenzen auf entfernte Objekte und die automatische Aktivierung der Objekte über diese Referenzen.

Schnittstellen

ActivationInstantiator	Methoden, um Objektgruppen zu erzeugen, die aktiviert werden können
ActivationMonitor	Methoden, um zu reagieren, wenn sich entweder ein Objekt einer Aktivierungsgruppe oder die Gruppe die Aktivität ändert

ActivationSystem	Methoden, um Gruppen aktivierbarer Objekte zu registrieren
Activator	Methoden, um entfernte Objekte zu aktivieren

Klassen

Activatable	Entfernte Objekte, die persistenten Zugriff über die Zeit benötigen und aktiviert werden können
ActivationDesc	Ein Descriptor mit Informationen, die zur Aktivierung von Objekten benötigt werden
ActivationGroup	Verhaltensweisen, um aktivierbare Objekte zu gruppieren und zu überwachen
ActivationGroupDesc	Descriptor mit Informationen, um Aktivierungsgruppen zu erzeugen und wiederherzustellen
ActivationGroupDesc. CommandEnvironment	Unterstützt die Implementierung von Defaultoptionen für ActivationGroup-Objekte
ActivationGroupID	Identifizierung einer registrierten Aktivierungsgruppe
ActivationID	Identifizierung für ein aktivierbares Objekt

java.rmi.dgc

Dieses Paket, das mit Java 1.1 eingeführt wurde, unterstützt Algorithmen für die verteilte Garbage-Collection.

Schnittstellen

DGC	Methoden zur Reinigung von Verbindungen bei ungenutzten Clients

Klassen

Lease	Enthält eine eindeutige VM-Kennung und eine Lease-Dauer
VMID	Methoden zur Pflege eindeutiger VMID auf allen virtuellen Java-Maschinen

java.rmi.registry

Dieses Paket wurde mit Java 1.1 eingeführt, um RMI-Registrierungen zu verarbeiten.

Schnittstellen

Registry	Klasse zum Ermitteln der Registrierung für unterschiedliche Hosts
RegistryHandler	Methoden, um auf eine Implementierung der Registry-Schnittstelle zuzugreifen (Java 1.2)
RegistryManager	Methoden zur Schnittstellenbildung zu Privatimplementierungen (nur Java 1.1)

Klassen

LocateRegistry	Dient zum Ermitteln der Bootstrap-Registrierung auf einem bestimmten Host

java.rmi.server

Dieses mit Java 1.1 eingeführte Paket kümmert sich um die Server-Seite bei RMI.

Schnittstellen

LoaderHandler	Methoden, um das entfernte Laden zu handhaben
RMIFailureHandler	Methoden zur Handhabung im Fall, daß die RMI-Laufzeitversion einen Socket oder ServerSocket nicht erstellen kann
RMIClientSocket-Factory	Zugriff auf Client Sockets für RMI Aufrufe (Java 1.2)
RMISServerSocket-Factory	Zugriff auf Server Sockets für RMI Aufrufe (Java 1.2)
RemoteCall	Methoden zur Implementierung von Aufrufen eines Remote-Objektes
RemoteRef	Stellt einen Handle für ein Remote-Objekt dar
ServerRef	Stellt den serverseitigen Handle für die Implementierung eines Remote-Objekts dar
Skeleton	Stellt eine serverseitige Entität dar, die Aufrufe der eigentlichen Remote-Objekt-Implementierung ausgibt
Unreferenced	Methoden zum Erhalt einer Benachrichtigung, wenn keine Remote-Verweise mehr vorliegen

Klassen

LogStream	Bietet einen Mechanismus zur Protokollierung von Fehlern, die für die Systemüberwacher von Interesse sein können
ObjID	Dient zur eindeutigen Identifizierung von Remote-Objekten in einer VM
Operation	Enthält eine Beschreibung einer Java-Methode
RMIClassLoader	Bietet statische Methoden für das Laden von Klassen über das Netz

RMISocketFactory	Wird von der RMI-Laufzeitversion benutzt, um Client- und Server-Sockets für RMI-Aufrufe zu ermitteln
RemoteObject	Liefert die Remote-Semantik von Object durch Implementieren von Methoden für hashCode(), equals() und toString()
RemoteServer	Superklasse für alle Server-Implementierungen, die den Rahmen für die Unterstützung einer breiten Palette von Remote-Referenzsemantiken liefert
RemoteStub	Stub-Objekte unterstützen exakt die gleiche Gruppe von Remote-Schnittstellen, die durch die tatsächliche Implementierung des Remote-Objekts definiert ist
UID	Abstraktion zum Erstellen von Bezeichnern, die in bezug auf den Host, auf dem die Generierung stattfand, eindeutig sind
UnicastRemoteObject	Definiert ein nicht repliziertes Remote-Objekt, dessen Referenzen nur während des aktiven Server-Prozesses gültig sind

java.security

Das Paket java.security enthält Klassen und Schnittstellen, die dem Programmierer die Implementierung von Zertifikaten und digitalen Unterschriften in Java-Komponenten ermöglichen.

Schnittstellen

Certificate	Methoden zum Verwalten eines Zertifikats einschließlich Ver- und Entschlüsselung (seit Java 1.2 als deprecated deklariert)
Guard	Methoden, um den Zugriff auf ein anderes Objekt zu schützen (Java 1.2)
Key	Methoden, um die Funktionalität, die von allen Key-Objekten geteilt wird, zu definieren (Java 1.2)
Principal	Darstellung der Hauptkomponente eines Zertifikats
PrivateKey	Methoden, um für alle Private-Key-Schnittstellen zu gruppieren und Typsicherheit zur Verfügung zu stellen (Java 1.2)
PrivilegedAction	Schnittstelle, um privilegierte Aktionen auszuführen, die keine checked exceptions werfen (Java 1.2)
Privileged-ExceptionAction	Schnittstelle, um privilegierte Aktionen auszuführen, die checked exceptions werfen (Java 1.2)
PublicKey	Methoden, um für alle Public-Key-Schnittstellen zu gruppieren und Typsicherheit zur Verfügung zu stellen (Java 1.2)

Klassen

AccessControlContext	Eine Umgebung, die verwendet wird, um Entscheidungen über die Zugriffskontrolle zu treffen (Java 1.2)

AccessController	Anbieter von kontrolliertem Zugriff auf Basis einer Sicherheitsrichtlinie (Java 1.2)
AlgorithmParameterGenerator	Verhaltensweisen, um mit einem bestimmten Algorithmus Parameter zu erzeugen (Java 1.2)
AlgorithmParameterGeneratorSpi	Service-Provider-Schnittstelle für AlgorithmParameterGenerator (Java 1.2)
AlgorithmParameters	Repräsentation kryptographischer Parameter (Java 1.2)
AlgorithmParametersSpi	Service-Provider-Schnittstelle für AlgorithmParameters (Java 1.2)
AllPermission	Enthält alle anderen Genehmigungen (Java 1.2)
BasicPermission	Verhaltensweisen, um elementare Genehmigungen für die Zugriffskontrolle anzubieten (Java 1.2)
CodeSource	URL-Erweiterung, um den Public-Key für die Echtheitsbestätigung einzubinden (Java 1.2)
DigestInputStream	Ein Eingabestream, der über einen Nachrichtenfingerabdruck verfügt
DigestOutputStream	Ein Ausgabestream, der über einen Nachrichtenfingerabdruck verfügt
GuardedObject	Ein Objekt, das zum Schutz des Zugriffs auf ein anderes Objekt verwendet wird (Java 1.2)
Identity	Methoden zur Verwaltung von Identitäten, die Objekte sein können, z.B. Leute, Firmen oder Organisationen, die zur Nutzung eines öffentlichen Schlüssels berechtigt sind (diese Klasse ist seit Java 1.2 als deprecated deklariert)
IdentityScope	Methoden zur Definition des Umfangs einer Identität einschließlich Name der Identität, ihr Schlüssel und zugehörige Zertifikate (diese Klasse ist seit Java 1.2 als deprecated deklariert)
Key	Abstrakte Klasse, die einen kryptographischen Schlüssel darstellt (nur Java 1.1)
KeyFactory	Konverter, der Schlüssel in eine Schlüsselspezifikation konvertiert (Java 1.2)
KeyFactorySpi	Service-Provider-Schnittstelle für KeyFactory-Objekte (Java 1.2)
KeyPair	Einfacher Halter eines Schlüsselpaares (ein öffentlicher und ein privater Schlüssel)
KeyPairGenerator	Generator zur Erzeugung eines Schlüsselpaares (Java 1.2)
KeyPairGeneratorSpi	Service-Provider-Schnittstelle für KeyPairGenerator-Objekte (Java 1.2)
KeyStore	Im Arbeitsspeicher verwaltete Sammlung von Keys und zugeordneten Zertifikaten (Java 1.2)

KeyStoreSpi	Bereitstellung eines Security Provider Interface (SPI) für die KeyStore- Klasse (Java 1.2)
MessageDigest	Methoden, die die Funktionalität eines Algorithmus für Nachrichtenfingerabdrücke bereitstellen
MessageDigestSpi	Service-Provider-Schnittstelle für MessageDigest-Objekte (Java 1.2)
Permission	Definiert ein Zugriffsrecht auf eine geschützte Quelle (Java 1.2)
Permission-Collection	Implementiert eine Gruppe von Permission-Objekten (Java 1.2)
Permissions	Heterogene Sammlung von Permission-Objekten (Java 1.2)
Policy	Repräsentation einer Richtlinie für die Java-Laufzeitumgebung (Java 1.2)
Provider	Repräsentiert einen Security Package Provider (SPP) für das JavaSecurity_API
SecureClassLoader	Erweiterung von ClassLoader für Klassen mit Code-Quellen und Signer Sets (Java 1.2)
SecureRandom	Erzeugt eine Zufallszahl
SecureRandomSpi	Service-Provider-Schnittstelle für die SecureRandom-Klasse (Java 1.2)
Security	Methoden zur Verwaltung von Security Package Providers (SPP) (die Methoden setParameter() und getParameter() wurden seit Java 1.2 als deprecated deklariert)
SecurityPermission	Verhaltensweisen zur Sicherheitszugriffskontrolle (Java 1.2)
Signature	Liefert den Algorithmus für digitale Unterschriften
SignatureSpi	Service-Provider-Schnittstelle für Signature-Objekte (Java 1.2; die Methoden engineGetParameter() und engineSetParameter() wurden seit Java 1.2 als deprecated deklariert)
SignedObject	Authentisches Laufzeitobjekt, dessen Integrität nicht zerstört werden kann, ohne daß dies erkannt wird (Java 1.2)
Signer	Stellt eine Identität dar, die auch unterzeichnen kann
Unresolved-Permission	Von Permission abgeleitete Klasse, die keine Permission-Klasse hat, auf die zugegriffen werden kann (Java 1.2)

java.security.acl

Das Paket java.security.acl enthält die Schnittstellen zur Datenstruktur, die den Zugriff auf Ressourcen überwacht.

Schnittstellen

Acl	Schnittstelle, die eine Access Control List (ACL) darstellt (eine Datenstruktur, die den Zugriff auf Ressourcen überwacht)
AclEntry	Methoden, die dem Programmierer das Hinzufügen, Entfernen oder Festlegen von Rechten für die Principals jedes AclEntry im ACL ermöglichen
Group	Methoden, die dem Programmierer das Hinzufügen oder Entfernen eines Mitglieds in der Gruppe der Principals ermöglichen
Owner	Stellt den Inhaber einer ACL dar
Permission	Diese Schnittstelle stellt die Art des gewährten Zugriffs auf eine Ressource dar, z.B. ein Principal im ACL

java.security.cert

Dieses Paket wurde mit Java 1.2 zur Zertifizierung der Identität eingeführt.

Schnittstellen

X509Extension	Erweiterungen, die für X.509-v3-Zertifikate und v2 Certificate-Revocation Lists definiert wurden

Klassen

Certificate	Identitätszertifikate mit unterschiedlichen Formaten und allgemeiner Verwendung
CertificateFactory	Methoden für die Erstellung von Zertifikaten und Certificate Revocation Lists
CertificateFactorySpi	Bereitstellung eines Security Provider Interface (SPI) für die CertificateFactory-Klasse
CRL	Implementierung einer Certificate Revocation List (CRL)
X509CRLEntry	Abstrakte Klasse für zurückgezogene Zertifikate in einer X509 CRL (Certificate Revocation List)
X509CRL	Abstrakte Klasse für eine X.509 Certificate Revocation List (CRL)
X509Certificate	Abstrakte Klasse für X.509-Zertifikate

java.security.interfaces

Schnittstellen

DSAKey	Methoden zur Authentifizierung von Komponenten einschließlich über das Web verteilter Steuerungen für Java-Applets und ActiveX
DSAKeyPairGenerator	Methoden, die von Objekten implementiert werden, die DSA-Schlüsselpaare generieren können
DSAParams	Methoden, die es Programmierern ermöglichen, Base, Prime und Sub-Prime zu erhalten
DSAPrivateKey	Schnittstelle zu einem privaten DSA-Schlüssel
DSAPublicKey	Schnittstelle zu einem öffentlichen DSA-Schlüssel
RSAPrivateCrtKey	Schnittstelle zu einem privaten RSA-Schlüssel mit Unterstützung für das »Chinese Remainder Theorem«
RSAPrivateKey	Schnittstelle zu einem privaten RSA-Schlüssel
RSAPublicKey	Schnittstelle zu einem öffentlichen RSA-Schlüssel

java.security.spec

Dieses Paket enthält Klassen und Schnittstellen mit Spezifikationen für kryptographische Schlüssel.

Schnittstellen

AlgorithmParameterSpec	Diese Schnittstelle stellt keine Konstanten oder Methoden zur Verfügung. Sie wird benutzt, um ein Objekt zu identifizieren, das Parameter für kryptographische Algorithmen zur Verfügung stellt
KeySpec	Diese Schnittstelle stellt keine Konstanten oder Methoden zur Verfügung. Sie wird benutzt, um ein Objekt zu identifizieren, das einen Schlüssel für einen kryptographischen Algorithmus darstellt

Klassen

DSAParameterSpec	Diese Klasse stellt Parameter zur Verfügung, um einen Digital Signature Algorithm (DSA) zu implementieren
DSAPrivateKeySpec	Implementierung eines privaten DSA-Schlüssels
DSAPublicKeySpec	Implementierung eines öffentlichen DSA-Schlüssels
EncodedKeySpec	Implementierung eines verschlüsselten öffentlichen oder privaten Schlüssels
PKCS8EncodedKeySpec	Diese Klasse repräsentiert die PKCS-#8-Standardverschlüsselung eines privaten Schlüssels
RSAPrivateCRTKeySpec	Spezifizierung eines privaten RSA-Schlüssels mit Hilfe von »Chinese Remainder Theorem«-Werten

RSAPrivateKeySpec	Implementierung eines privaten RSA-Schlüssels
RSAPublicKeySpec	Implementierung eines öffentlichen RSA-Schlüssels
X509EncodedKeySpec	Implementierung der X.509 Standardverschlüsselung eines privaten oder öffentlichen Schlüssels

java.sql

Das Paket java.sql enthält Klassen, Schnittstellen und Methoden zur Verbindung von Java-Applikationen mit Backend-Datenbanken.

Schnittstellen

Array	Referenz auf ein Array, das auf dem Datenbankserver gespeichert ist (Java 1.2)
Blob	Referenz auf ein Binary Large Object, das auf dem Datenbankserver gespeichert ist (Java 1.2)
CallableStatement	Methoden zur Ausführung gespeicherter Prozeduren und Handhabung mehrerer ResultSet-Objekte
Clob	Referenz auf ein Character Large Object, das auf dem Datenbankserver gespeichert ist (Java 1.2)
Connection	Stellt eine Session mit der Datenbank dar
DatabaseMetaData	Schnittstelle, mit der Programmierer auf hoher Ebene Informationen über die Datenbank erhalten können
Driver	Methoden zum Verbinden mit einer Datenbank
PreparedStatement	Methoden zum Starten zuvor kompilierter SQL-Anweisungen
Ref	Referenz auf einen gespeicherten SQL-Wert (Java 1.2)
ResultSet	Methoden zum Abrufen von Werten und zum Ausführen von SQL-Anweisungen
ResultSetMetaData	Methoden, die Informationen über den Typ und die Eigenschaften der Spalten in einem ResultSet liefern
SQLData	Unterstützung bei der Abbildung von SQL-Datentypen auf Java-Datentypen (Java 1.2)
SQLInput	Schnittstelle, die einen Eingabestrom aus einer SQL-UDT-Instanz repräsentiert (Java 1.2)
SQLOutput	Schnittstelle, die einen SQL-UDT-Ausgabestrom repräsentiert (Java 1.2)
Statement	Wird für statische SQL-Anweisungen verwendet
Struct	Schnittstelle, die einen strukturierten SQL-Datentypen kapselt (Java 1.2)

Klassen

Date	Liefert Methoden zur Formatierung von und zum Verweisen auf Datumswerte(n)
DriverManager	Ermöglicht die Verwaltung eines Satzes von JDBC-Treibern
DriverPropertyInfo	Liefert Methoden zum Ermitteln verschiedener Eigenschaften eines Treibers
Time	Liefert Methoden zur Formatierung von und zum Verweisen auf Zeitwerte(n)
Timestamp	Wrapper, der den SQL-TIMESTAMP-Wert enthält
Types	Definiert Konstanten, die zur Identifizierung von SQL-Typen verwendet werden

java.text

Das Paket `java.text` enthält Klassen und Methoden zur Formatierung von Objekten wie Zahlen, Datumsangaben, Zeitangaben usw. in einer Zeichenkette oder zum Übergeben eines Strings an ein anderes Objekt, z.B. eine Zahl, Datum, Zeit usw.

Schnittstellen

AttributedCharacterIterator	Erweitert die CharacterIterator-Schnittstelle, um durch Text, der Formatierungs- oder andere Attribute hat, iterieren zu können.
CharacterIterator	Methoden zur Übergabe einer Zeichenkette und Rückgabe diverser Informationen über diese Zeichenkette

Klassen

Annotation	Methoden, um mit Textattributwerten zu arbeiten (Java 1.2)
AttributedCharacter-Iterator.Attribute	Definition von Attributschlüsseln, die benutzt werden, um Textattribute zu identifizieren (Java 1.2)
AttributedString	Kapselung von Text und darauf bezogenen Attributs-Informationen (Java 1.2)
BreakIterator	Unterstützung beim Finden von Textumbruchstellen (Java 1.2)
ChoiceFormat	Methoden, die es ermöglichen, Zahlen Zeichenketten zuzuordnen
CollatedString	Liefert eine Möglichkeit zur Verwendung internationaler Zeichenketten in einer Hash-Tabelle oder sortierten Auflistung (nur Java 1.1)
Collation	Ermöglicht den Vergleich von Unicode-Text (nur Java 1.1)

CollationElementIterator	Methoden, die es erlauben, Sortierschlüsselzeichen aus Zeichen einer Zeichenkette zu erzeugen
CollationKey	Wird benutzt, um zwei Collator-Objekte zu vergleichen (Java 1.2)
Collator	Unterstützt landesspezifische Stringvergleiche (Java 1.2)
DateFormat	Abstrakte Klasse, die verschiedene Subklassen zur Datum-/Zeit-Formatierung enthält
DateFormatData	Methoden zur Festlegung der Datum-/Zeit-Formatierungsdaten (nur Java 1.1)
DateFormatSymbols	Methoden zur Festlegung der landesspezifischen Datum-/Zeit-Formatierungsdaten (Java 1.2)
DecimalFormat	Methoden zum Formatieren von Zahlen
DecimalFormatSymbols	Methoden zur Festlegung der landesspezifischen Dezimalstellen-Formatierungsdaten (Java 1.2)
FieldPosition	Wird benutzt, um Felder in formatierten Ausgaben zu identifizieren (Java 1.2)
Format	Basisklasse für alle Formate
FormatStatus	Dient zum Ausrichten formatierter Objekte (nur Java 1.1)
MessageFormat	Methoden zur Erstellung verketteter Nachrichten.
NumberFormat	Abstrakte Klasse für alle Zahlenformate; enthält Subklassen mit Methoden zur Formatierung und Übergabe von Zahlen
NumberFormatData	Kapselt lokalisierbare Zahlenformatierungsdaten (nur Java 1.1)
ParsePosition	Wird benutzt, um die aktuelle Parse-Position zu verfolgen (Java 1.2)
ParseStatus	Ermittelt den Status der Übergabe bei der Übergabe einer Zeichenkette mit unterschiedlichen Formaten (nur Java 1.1)
RuleBasedCollator	Unterstützt das Sortieren nach bestimmten Regeln (Java 1.2)
SimpleDateFormat	Methoden zum Formatieren eines Datums oder einer Zeit in einer Zeichenkette
SortKey	Methoden zum bitweisen Vergleichen von Zeichenketten (nur Java 1.1)
StringCharacterIterator	Methoden zum bidirektionalen Iterieren durch eine Zeichenkette
TableCollation	Implementiert Collation mittels datengesteuerter Tabellen (nur Java 1.1)
TextBoundary	Dient zur Ermittlung von Grenzen in vorgegebenem Text (nur Java 1.1)

java.util

Das Paket java.util enthält verschiedene Utility-Klassen und Schnittstellen, darunter Zufallszahlen, Systemeigenschaften und andere nützliche Klassen.

Schnittstellen

Collection	Methoden, um mit willkürlichen Zusammenstellungen von Objekten zu arbeiten (Java 1.2)
Comparator	Methoden, um eine Vergleichsfunktion zu implementieren (Java 1.2)
Enumeration	Methoden zur Auflistung von Werten
EventListener	Methoden für die Ereignishandhabung (Java 1.1)
Iterator	Methoden, um über eine geordnete Liste zu iterieren (Java 1.2)
List	Methoden, um mit einer geordneten Liste von Objekten zu arbeiten (Java 1.2)
ListIterator	Unterstützt das Iterieren über ein List-Objekt (Java 1.2)
Map	Methoden für die Zuordnung zwischen zwei Objektgruppen (Java 1.2)
Map.Entry	Methoden für ein einzelnes Zuordnungselement (Java 1.2)
Observer	Methoden zur Aktivierung der Observierung von Observable-Objekten
Set	Implementierung einer Gruppe, in der jedes Element nur einmal vorhanden ist (Java 1.2)
SortedMap	Bestimmung einer Ordnung zwischen den Zuordnungselementen (Java 1.2)
SortedSet	Bestimmung einer Ordnung zwischen den einzelnen Elementen eines Sets (Java 1.2)

Klassen

AbstractCollection	Abstrakte Implementierung der Collection-Schnittstelle (Java 1.2)
AbstractList	Abstrakte Implementierung der List-Schnittstelle (Java 1.2)
AbstractMap	Abstrakte Implementierung der Map-Schnittstelle (Java 1.2)
AbstractSequentialList	Stellt einen Datenspeicher mit sequentiellm Zugriff zur Verfügung (Java 1.2)
AbstractSet	Abstrakte Implementierung der Set-Schnittstelle (Java 1.2)
ArrayList	Von AbstractList abgeleitete Klasse, die in Form eines Arrays implementiert ist (Java 1.2)
Arrays	Methoden, um Arrays zu bearbeiten (Java 1.2)
BitSet	Eine Bit-Menge
Calendar	Allgemeiner Kalender (Java 1.1)
Collections	Statische Methoden, um mit Gruppen von Objekten zu arbeiten (Java 1.2)

Date	Aktuelles Systemdatum sowie Methoden zum Erzeugen und Abgleichen von Datumsangaben
Dictionary	Abstrakte Klassen, die Schlüssel und Werte einander zuordnet (Superklasse von Hashtable)
EventObject	Ereignisobjekt, das mit einem anderen Objekt verbunden ist (Java 1.1)
GregorianCalendar	Gregorianischer Kalender, d.h. der Kalender, den Sie wahrscheinlich einsetzen (Java 1.1)
HashMap	Implementierung der Map-Schnittstelle, die eine Hash-Tabelle benutzt (Java 1.2)
HashSet	Implementierung der Set-Schnittstelle, die eine Hash-Tabelle benutzt (Java 1.2)
Hashtable	Eine Hash-Tabelle
LinkedList	Kapselung einer Datenstruktur in Form einer verketteten Liste (Java 1.2)
ListResourceBundle	Ressourcenunterstützung für ein Locale-Objekt (Java 1.1)
Locale	Beschreibung eines geographischen Ortes (Java 1.1)
Observable	Abstrakte Klasse für überwachbare Objekte
Properties	Hash-Tabelle, die Eigenschaften zum Setzen und Abrufen persistenter Merkmale des Systems oder einer Klasse enthält
PropertyPermission	Implementiert eine Zugriffskontrolle auf Systemeinstellungen (Java 1.2)
PropertyResourceBundle	Die Objekte dieser Klasse stellen Eigenschaften aus einer Datei heraus zur Verfügung (Java 1.1)
Random	Utilities zur Erstellung von Zufallszahlen
ResourceBundle	Gruppe von Objekten, die auf ein Locale-Objekt bezogen sind (Java 1.1)
SimpleTimeZone	Vereinfachte Zeitzone (Java 1.1)
Stack	Implementiert einen Stack (eine Warteschlange mit dem Prinzip »Last in, first out«)
StringTokenizer	Utilities zum Aufteilen von Zeichenketten in einzelne »Tokens«
TimeZone	Allgemeine Zeitzone (Java 1.1)
TreeMap	Implementierung der Map-Schnittstelle, die einen Baum benutzt (Java 1.2)
TreeSet	Implementierung der Set-Schnittstelle, die einen Baum benutzt (Java 1.2)
Vector	Array von Objekten, dessen Größe dynamisch verändert werden kann
WeakHashMap	Implementierung der Map-Schnittstelle, um eine Hashtable-basierte Map mit schwachen Keys zur Verfügung zu stellen (Java 1.2)

java.util.zip

Das Paket java.util.zip bietet Klassen zur Handhabung von ZIP- und GZIP-Dateien.

Schnittstellen

Checksum	Methoden zur Prüfsummenberechnung

Klassen

Adler32	Berechnung einer Adler-32-Prüfsumme
CRC32	Berechnung einer CRC-32-Prüfsumme
CheckedInputStream	Eingabestream mit zugehöriger Prüfsumme
CheckedOutputStream	Ausgabestream mit zugehöriger Prüfsumme
Deflater	Dient der Komprimierung nicht komprimierter Dateien
DeflaterOutputStream	Ausgabestream, der komprimiert
GZIPInputSteam	Eingabestream aus einer GZIP-Datei
GZIPOutputStream	Ausgabestream in eine GZIP-Datei
Inflater	Dient der Dekomprimierung komprimierter Dateien
InflaterInputStream	Eingabestrom, der dekomprimiert
ZipEntry	Dateieintrag innerhalb einer ZIP-Datei
ZipFile	Komplette ZIP-Datei
ZipInputStream	Eingabestream aus einer ZIP-Datei
ZipOutputStream	Ausgabestrom in eine ZIP-Datei

javax.accessibility

Dieses Paket, das mit Java 1.2 eingeführt wurde, erweitert die Funktionalität der Benutzerschnittstelle eines Programms, so daß diese von Technologien wie Braille-Terminals, Bildschirmleser und Spracherkennung verwendet werden kann.

Schnittstellen

Accessible	Methoden, um ein Element der Benutzerschnittstelle besser zugänglich zu machen
AccessibleAction	Methoden, um herauszufinden, welche Aktionen von einer Komponente unterstützt werden. Zusätzlich auch Methoden, um auf diese Aktionen zuzugreifen

AccessibleComponent	Methoden, um das Verhalten und Aussehen von GUI-Komponenten, die dies unterstützen, festzulegen
AccessibleHypertext	Dieses Interface wird von GUI-Komponenten implementiert, die Hypertext anzeigen. Es vereinfacht die Hypertextanzeige
AccessibleSelection	Methoden, um zu bestimmen, welche Unterkomponenten einer GUI-Komponente selektiert wurden, und um den Selektionsstatus dieser Komponenten zu beeinflussen
AccessibleText	Methoden, um Text, der an einer bestimmten Position auf einem grafischen Ausgabegerät angezeigt wird, in bezug auf Inhalt, Attribute und Layout zu beeinflussen
AccessibleValue	Methoden, um numerische Werte zu setzen und auszulesen und den Wertebereich zu bestimmen

Klassen

AccessibleBundle	Diese Klasse gibt Zugriff auf die resource bundles und unterstützt Zeichenkettenumwandlung
AccessibleContext	Dies ist eine Kern-Klasse der accessibility-API und ermöglicht den Zugriff auf andere Objekte. Sie definiert die Daten, die von allen Accessible-Objekten verwendet werden. Klassen, die die assistive technology implementieren, werden von dieser Klasse abgeleitet
AccessibleHyperlink	Diese Klasse ermöglicht den Einsatz der Accessibility-Klassen für Hyperlinks und Gruppen von Hyperlinks
AccessibleResourceBundle	Diese Klasse ist von java.util.ListResourceBundle abgeleitet, die ein resource bundle für Assistive-technology-Applikationen implementiert
AccessibleRole	Eine präzise Beschreibung der Rolle, die ein Element in der Benutzerschnittstelle spielt
AccessibleState	Eine präzise Beschreibung des aktuellen Status eines Elementes in der Benutzerschnittstelle
AccessibleStateSet	Der Satz aller AccessibleState-Objekte. Dieser repräsentiert den Gesamtstatus eines Elementes der Benutzerschnittstelle

javax.swing

Dieses Paket wurde mit Java 1.2 eingeführt. Es bietet einen neuen Satz von Komponenten für grafische Benutzerschnittstellen und andere Verbesserungen für die Benutzerschnittstelle. Swing-Komponenten können automatisch das Look and Feel einer Plattform annehmen (z.B. Windows 95, Macintosh und Solaris). Die Klassen in diesem Paket verwenden automatisch ein spezielles Paket für das gewählte Look and Feel, darunter javax.swing.plaf, javax.swing.plaf.metal, javax.swing.plaf.multi und javax.swing.plaf.basic.

Die Swing-Komponenten umfassen Duplikate aller AWT-Komponenten von Java 1.1 und viele zusätzliche Komponenten. Es setzt sich aus dem javax.swing-Paket und den folgenden Subpaketen

zusammen:

- javax.swing
- javax.swing.border
- javax.swing.colorchooser
- javax.swing.event
- javax.swing.filechooser
- javax.swing.plaf
- javax.swing.plaf.basic
- javax.swing.plaf.metal
- javax.swing.plaf.multi
- javax.swing.table
- javax.swing.text
- javax.swing.text.html
- javax.swing.tree
- javax.swing.undo

Schnittstellen

Action	Methoden, um unterschiedlichen Komponenten die Action-command-Funktionalität zu geben
BoundedRangeModel	Methoden, um die Daten in einer Bildlaufleiste oder einem Schieberegler zu repräsentieren
ButtonModel	Methoden, die den Zustand einer Schaltfläche repräsentieren
CellEditor	Methoden, um Tabellen in Komponenten, wie z.B. Listenfeldern oder Kombinationslistenfeldern, zu bearbeiten
ComboBoxEditor	Methoden, um Kombinationslistenfelder zu bearbeiten
ComboBoxModel	Methoden, um die Daten in einem Kombinationslistenfeld zu repräsentieren
DesktopManager	Methoden, um das Look-and-Feel-Verhalten für JDesktopPane zu implementieren
Icon	Kleine Bildchen, die auf Komponenten verwendet werden
JcomboBox.KeySelectionManager	Methoden, um einen KeySelectionManager zu definieren
ListCellRenderer	Methoden, um die Zellen in einer Jlist zu zeichnen
ListModel	Methoden, um die Daten in einer Liste zu repräsentieren
ListSelectionModel	Methoden, um die aktuelle Auswahl in einer Liste zu repräsentieren
MenuElement	Methoden, die von Objekten implementiert werden, die in einem Menü platziert werden
MutableComboBox-Model	Erweiterung der ComboBoxModel-Schnittstelle, die auch Updates unterstützt

Renderer	Methoden, um Zugriff auf GUI Komponenten zu erhalten und ihre Einstellungen zu ändern
RootPaneContainer	Methoden, die von Fensterkomponenten der obersten Hierarchieebene implementiert werden
ScrollPaneConstants	Konstanten für die Verwendung mit JScrollPane-Objekten
Scrollable	Methoden, um Informationen für einen scrollbaren Container anzubieten
SingleSelectionModel	Methoden, um die einzig mögliche Auswahl in einer Liste zu repräsentieren
SwingConstants	Konstanten für die Verwendung mit Swing-Komponenten
UIDefaults.ActiveValue	Methoden, um einen Eintrag in der Standardtabelle zu speichern
UIDefaults.LazyValue	Methoden, um einen Eintrag in der Standardtabelle zu speichern, der nicht aufgebaut ist, bis er nicht angefordert wird
WindowConstants	Konstanten für die Verwendung mit JWindow-Komponenten

Klassen

AbstractAction	Verhaltensweisen für Aktionsobjekte
AbstractButton	Gemeinsame Verhaltensweisen für Schaltflächenkomponenten (die Methoden getLabel() und setLabel() wurden mit Java 1.2 als deprecated deklariert)
AbstractListModel	Datenmodell, das eine Listen-Komponente füllt
BorderFactory	Erzeuger für Standardumrandungsobjekte
Box	Ein Container, der den Box-Layout-Manager verwendet
Box.Filler	Box-Container ohne Ansicht
BoxLayout	Box-Layout-Manager
ButtonGroup	Klasse, die in einer Gruppe von Schaltflächen nur die Auswahl einer einzigen Schaltfläche zulässt
CellRendererPane	Klasse, die zwischen der Darstellung einer Zelle und der Komponente sitzt, um Aufrufe von repaint() und invalidate() abzublocken
DebugGraphics	Unterstützung für das Debuggen
DefaultBoundedRangeModel	Generisches Modell eines gebundenen Bereiches
DefaultButtonModel	Standardversion des Datenmodells einer Schaltfläche
DefaultCellEditor	Standardeditor für Zellen von Tabellen und Baumansichten
DefaultComboBox-Model	Standardversion des Datenmodells einer Combobox
DefaultDesktopManager	Generischer Desktopmanager
DefaultFocusManager	Generischer Fokusmanager

DefaultListCell-Renderer	Standarddarstellung für eine List cell
DefaultListCell-Renderer.UIResource	Innere Klasse von DefaultListCellRenderer, die die UIResource-Schnittstelle implementiert
DefaultListModel	Standardversion des Datenmodells einer Listenkompone
DefaultListSelectionModel	Standardversion der ausgewählten Elemente in einer Liste
DefaultSingleSelectionModel	Standardversion des einzigen ausgewählten Elementes in einer Liste
FocusManager	Der Fokusmanager einer Komponente
GrayFilter	Bild, das eine Komponente deaktiviert, indem es diese grau darstellt
ImageIcon	Implementierung der Icon-Schnittstelle
JApplet	Swing-Applet
JButton	Swing-Schaltfläche
JCheckBox	Swing-Kontrollkästchen
JCheckBoxMenuItem	Swing-Version der umschaltbaren Menüeinträge
JColorChooser	Swing-Farbauswahl-Dialog
JComboBox	Swing-Kombinationslistenfeld
JComponent	Basisklasse aller Swing-Komponenten
JDesktopPane	Swing-Desktop-Bereich
JDialog	Swing-Dialogfenster
JEditorPane	Swing-Editorbereich
JFileChooser	Swing-Dateiauswahldialog
JFrame	Swing-Frame
JInternalFrame	Swing-interner Frame
JInternalFrame. JDesktopIcon	Icon für die Benutzung mit dem JInternalFrame-Objekt
JLabel	Swing-Label
JLayeredPane	Swing-Bereichscontainer mit mehreren Ebenen
JList	Swing-Listenfeld
JMenu	Swing-Menü
JMenuBar	Swing-Menüleiste
JMenuItem	Swing-Menüeintrag
JOptionPane	Swing-Standarddialoge
JPanel	Swing-Panel-Container
JPasswordField	Swing-Textfeld, in dem die eingegebenen Zeichen durch andere Zeichen verdeckt werden (zur Eingabe von Kennwörtern)
JPopupMenu	Swing-Kontextmenü

JPopupMenu.Separator	Swing-Kontextmenütrennlinie
JProgressBar	Swing-Fortschrittsanzeige
JRadioButton	Swing-Optionsfeld
JRadioButtonMenuItem	Swing-Optionsfeld in einem Menü
JRootPane	Swing-Stammbereichscontainer
JScrollBar	Swing-Bildlaufleiste
JScrollPane	Swing-Version eines scrollbaren Bereichscontainers
JSeparator	Swing-Version einer Trennlinie für Menüs
JSlider	Swing-Schieberegler
JSplitPane	Swing-Bereichscontainer mit teilbarem Bereich
JTabbedPane	Swing-Bereichscontainer mit Registern
JTable	Swing-Tabelle
JTextArea	Swing-Version eines mehrzeiligen Textfeldes
TextField	Swing-Version eines einzeiligen Textfeldes
JTextPane	Swing-Textbereichscontainer
JToggleButton	Swing-Version einer umschaltbaren Schaltfläche
JToggleButton.Toggle-ButtonModel	Konfiguration von umschaltbaren Schaltflächen
JToolBar	Swing-Werkzeugleiste
JToolBar.Separator	Trennlinie für Werkzeugleisten
JToolTip	ToolTip-Komponente
JTree	Swing-Verzeichnisbaum
JTree.DynamicUtil-TreeNode	Dynamische Verwaltung von Baumknoten
JTree.EmptySelection-Model	Selektion von Bäumen
JViewport	Swing-Viewport-Container
JWindow	Swing-Fenstercontainer
KeyStroke	Eine Taste der Tastatur, die gedrückt wurde
LookAndFeel	Einstellbares Look and Feel
MenuSelection-Manager	Verwaltung der Auswahl von Menüpunkten
OverlayLayout	Overlay-Layout-Manager
ProgressMonitor	Verhaltensweisen, um den Fortschritt einer Operation zu verfolgen
ProgressMonitorInputStream	Verhaltensweisen, um den Fortschritt eines Datenstroms zu verfolgen
RepaintManager	Manager für die Anzeige von Komponenten
ScrollPaneLayout	Scroll-Pane-Layout-Manager
ScrollPaneLayout. UIResource	Zugriff auf eine UIResource

SizeRequirements	Größenmeßwerte, die von Layout-Managern verwendet werden
SwingUtilities	Utility-Methoden für Swing
Timer	Ereignisse werden in festgelegten Intervallen ausgelöst
ToolTipManager	Manager für ToolTip-Komponenten
UIDefaults	Standardwerte für Swing-Komponenten
UIManager	Klasse, um das aktuelle Look and Feel im Auge zu behalten
UIManager.LookAnd-FeelInfo	Speicherung von Look-and-Feel-Informationen
ViewportLayout	Viewport-Layout-Manager

javax.swing.event

Dies ist ein Subpaket von Swing, das mit Java 1.2 eingeführt wurde, um neue Ereignisse, die von Swing-Komponenten ausgelöst werden, zu unterstützen.

Schnittstellen

AncestorListener	Methoden, um eine Änderung in einer Swing-Komponente oder deren Superklasse anzumerken
CaretListener	Methoden, um auf einen CaretEvent zu reagieren
CellEditorListener	Unterstützt das Editieren von Tabellenzellen, indem auf Textänderungen reagiert wird
ChangeListener	Methoden, um auf Textänderungen zu reagieren
DocumentEvent	Methoden, um auf Änderungen in Dokumenten zu reagieren
DocumentEvent. ElementChange	Methoden, um auf Änderungen an einem Dokumentsymbol zu reagieren
DocumentListener	Methoden, um auf Änderungen in Dokumenten zu reagieren
HyperlinkListener	Methoden, um auf Änderungen an Hyperlinks zu reagieren
InternalFrameListener	Methoden, um auf den InternalFrameEvent zu reagieren
ListDataListener	Methoden, um auf den ListDataEvent zu reagieren
ListSelectionListener	Methoden, um auf Selektierungen in Listen zu reagieren
MenuDragMouse-Listener	Methoden, um auf den MenuDragMouseEvent zu reagieren
MenuKeyListener	Methoden, um auf das Drücken eines Menü-Shortcuts zu reagieren
MenuListener	Methoden, um auf Änderungen an Menüs zu reagieren
MouseListener	Methoden, um auf alle Maus-Events zu reagieren
PopupMenuListener	Methoden, um auf Änderungen an Popupmenüs zu reagieren

TableColumnModelListener	Methoden, um auf Änderungen in einer Tabellenspalte zu reagieren
TableModelListener	Methoden, um auf Änderungen in einer Tabelle zu reagieren
TreeExpansionListener	Methoden, um auf das Öffnen eines Zweiges eines Verzeichnisbaumes zu reagieren
TreeModelListener	Methoden, um auf Änderungen in einem Verzeichnisbaum zu reagieren
TreeSelectionListener	Methoden, um auf Auswahlen in einem Verzeichnisbaum zu reagieren
TreeWillExpand-Listener	Methoden, um auf das Aufklappen von Bäumen zu reagieren
UndoableEditListener	Methoden, um auf den UndoableEditEvent zu reagieren

Klassen

AncestorEvent	Ereignisse, wenn sich in einem JComponent-Objekt oder einem Objekt von deren Superklasse etwas ändert
CaretEvent	Ereignisse für Text-caret-Änderungen
ChangeEvent	Ereignisse für Textänderungen
DocumentEvent.Event-Type	Objekt, um alle Typen von Dokument-Ereignissen aufzuzählen
EventListenerList	Liste der Ereignis-Listener
HyperlinkEvent	Ereignisse für Hyperlinks
HyperlinkEvent. EventType	Objekt, um alle Typen von Hyperlink-Ereignissen aufzuzählen
InternalFrameAdapter	Standardimplementierung der InternalFrameListener-Schnittstelle
InternalFrameEvent	Stellt Ereignisse, die von javax.swing.JInternalFrame-Objekten stammen, zur Verfügung
ListDataEvent	Ereignisse für Änderungen in Komponenten vom Typ Liste
ListSelectionEvent	Ereignisse, wenn in einer Liste etwas ausgewählt wird
MenuDragMouseEvent	Ereignisse für menübezogene Drag-und-Drop-Operationen
MenuEvent	Menüereignisse
MenuKeyEvent	Auf Menüs bezogene Tastenereignisse
MouseInputAdapter	Standardimplementierung der MouseInputListener-Schnittstelle
PopupMenuEvent	Popupmenüereignisse
SwingPropertyChange-Support	Abgeleitet von java.beans.PropertyChangeSupport, um Swing Unterstützung zu geben
TableColumnModelEvent	Ereignisse bei Änderungen in einer Tabellenspalte
TableModelEvent	Ereignisse bei Änderungen in Tabellen
TreeExpansionEvent	Ereignisse beim Öffnen eines Zweiges in einer Verzeichnisan sicht
TreeModelEvent	Ereignisse beim Ändern von Einträgen in einer Verzeichnisan sicht

TreeSelectionEvent	Ereignisse bei Auswahlen in einer Verzeichnisansicht
UndoableEditEvent	Ereignis, daß eine Operation durchgeführt wurde, die zurückgenommen werden kann

javax.swing.undo

Dies ist ein Subpaket von Swing, das mit Java 1.2 eingeführt wurde und die Unterstützung von Rückgängig-Kommandos bietet.

Schnittstellen

StateEditable	Methoden für Objekte, deren Status rückgängig gemacht werden kann
UndoableEdit	Methoden, um eine vollständige Bearbeitung zu repräsentieren, die rückgängig gemacht werden kann

Klassen

AbstractUndoableEdit	Verhaltensweisen, die die UndoableEdit implementieren
CompoundEdit	Verhaltensweisen, um verschiedene Bearbeitungsschritte, die sich zurücknehmen lassen, in einem einzelnen, größeren zu kombinieren
StateEdit	Allgemeine Bearbeitung für Objekte, die Ihren Status ändern können
UndoManager	Listener für Ereignisse, die sich auf Bearbeitungen, die zurückgenommen werden können, beziehen
UndoableEditSupport	Verwaltung von Veränderungen, die zurückgenommen werden können

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Tag C

Java-Ressourcen im Internet

Dieser Anhang führt Bücher, World Wide Websites, Internet-Diskussions-Foren und andere Ressourcen auf, die Sie zur Erweiterung Ihres Java-Wissens nutzen können.

Die Website zur amerikanischen Ausgabe des Buches

Rogers Cadenhead, der Koautor des Buches, pflegt die offizielle Website zur amerikanischen Ausgabe dieses Buches unter der folgenden Adresse:

<http://www.prefect.com/java21>

Auf dieser Site finden Sie den gesamten Quelltext aller Projekte des Buches, eine Liste der bekannten Fehler und Lösungen (bezieht sich nur auf die amerikanische Ausgabe), aktualisierte Links und andere Informationen.

Andere Bücher

Neben diesem Buch sind beim Markt&Technik-Verlag noch einige andere Bücher erschienen, die Ihnen helfen, Ihr Wissen zum Thema Java und verwandten Themen zu erweitern. Das gesamte Verlagsprogramm können Sie auf der Website des Verlages einsehen unter <http://www.mut.de>.

Die JavaSoft-Site

Wie Sie bereits am ersten Tag erfahren haben, unterhält JavaSoft eine umfassende Website unter der Adresse:

<http://java.sun.com>

JavaSoft ist der Bereich von Sun Microsystems, der die neuen Versionen des Java Developer's Kit, die Java-Klassenbibliothek und andere Produkte produziert. Die JavaSoft-Site ist der erste Ort, an dem man sich wegen Informationen zu Java umsehen sollte. Neue Versionen des JDK und viele andere Programmierressourcen sind hier verfügbar.

Zum Zeitpunkt, als dieses Buch entstand, gliederte sich die JavaSoft-Site in die folgenden Bereiche:

- *What's New* - Ankündigungen neuer Produktreleases und Events, wie z.B. JavaOne (eine halbjährlich stattfindende Konferenz für Java-Programmierer). Dieser Bereich beinhaltet auch die Pressemitteilungen von JavaSoft und Terminpläne für Seminare, die von der Firma angeboten werden.
- *Read About Java* - Artikel über die Sprache. Diese richten sich an Leute, die sich zum erstenmal mit Java beschäftigen. Es lohnt sich, diesen Bereich einmal genauer anzusehen, da er sich explizit an Einsteiger richtet.

- *Products & APIs* - Alle Produkte und Dokumentationen, die von JavaSoft heruntergeladen werden können. Darunter auch das JDK, die Dokumentation der Sprache und andere Dateien.
- *Applets* - Eine Vorführung von Java-Programmen, die im Web laufen. Darunter befinden sich auch mehr als zwei Dutzend, die Sie für die Verwendung auf der eigenen Website anpassen können. Zusätzlich befinden sich hier auch Links zu einigen Applet-Archiven im Internet, darunter Gamelan von Developer.Com (<http://www.gamelan.com>) und der Java Applet Rating Service (JARS) unter <http://www.jars.com>.
- *For Developers* - Startpunkt zur Java Developer Connection. Hier finden Sie die gesammelten Ressourcen für den offiziellen Support von JavaSoft und die Dokumentation zu der Sprache Java im HTML-Format. Sie finden hier außerdem Informationen zu Konferenzen zu der Sprache, offizielle Java-Bücher und andere Ressourcen.
- *Java in the Real World* - Beiträge und Zeitpläne von »Erfolgsgeschichten« über die Sprache an Arbeitsplätzen und sonstigen Einsatzorten, die so unterschiedlich sind, wie die Post der USA, das UCLA Medical Center, Ergon Informatik und die Mars-Pathfinder-Mission.
- *Business & Licensing* - Bietet Richtlinien zur Lizenzierung und zu Warenzeichen- Fragen in bezug auf die Anwendung von Java-Produkten.
- *Support & Services* - Der Bereich für technischen und Kundensupport sowie für Verkaufsinformationen.
- *Marketing* - Detaillierte Informationen zu dem Programm »100 Percent Pure Java«, Java-Lösungen und andere Marketing- und Support-Bemühungen, die sich auf die Sprache beziehen.
- *Employment* - Liste der offenen Stellen bei JavaSoft in den Bereichen Engineering, technische Dokumentation, Marketing/Vertrieb und anderen Abteilungen.
- *Java Store* - Ein Katalog der offiziellen Merchandising-Produkte, die über das Web bestellt werden können. Darunter befinden sich z.B. Hemden, Kaffeetassen, T-Shirts und Baseball-Kappen.

Diese Site wird ständig mit nützlichen Ressourcen für Java-Programmierer aktualisiert. Etwas, wovon Sie sofort einen Nutzen haben, ist die Dokumentationsseite unter der folgenden Adresse:

<http://java.sun.com/docs/index.html>

Andere Websites

Da das Java-Phänomen zu einem großen Teil durch die Verwendung der Sprache auf Webseiten inspiriert wurde, konzentrieren sich eine große Anzahl von Sites auf Java und die Java-Programmierung.

Seiten zu Java-Büchern

Um einen Überblick über die vielen hundert Bücher, die zum Thema Java veröffentlicht wurden, zu bekommen, wurden einige Sites mit Buchbesprechungen zusammengestellt. Dort finden Sie Listen neuer, aktueller und nicht mehr gedruckter Bücher.

JavaWorld, ein Online-Magazin, das sich der Sprache und den Technologien im Umfeld widmet, pflegt eine Liste aktueller Java-Bücher und Java-Bücher, die bald erscheinen werden. Sie finden diese Liste unter der folgenden Adresse:

<http://www.javaworld.com/javaworld/books/jw-books-index.html>

Eine andere Übersicht Java-bezogener Bücher wird von Elliotte Rusty Harold herausgegeben, dem Autor vieler der Bücher, die auf der Webseite beschrieben werden. Harolds Liste, die Besprechungen zu vielen der Bücher beinhaltet, ist auf der folgenden Seite verfügbar:

<http://sunsite.unc.edu/javafaq/books.html>

Gamelans Java-Verzeichnis

Da Java eine objektorientierte Programmiersprache ist, ist es einfach, Ressourcen, die von anderen Programmierern geschaffen wurden, in eigenen Programmen zu verwenden. Bevor Sie mit einem wichtigen Java-Projekt beginnen, sollten Sie im World Wide Web nach Ressourcen suchen, die Sie eventuell in Ihrem Programm verwenden können.

Der Ausgangspunkt hierfür ist Gamelan von Developers.Com, die Website, die Java- Programme, Programmierressourcen und andere Informationen katalogisiert. Besuchen Sie die folgende Adresse:

<http://www.gamelan.com>

Gamelan ist das umfassendste Verzeichnis dieser Art. Dabei übertrifft es sogar JavaSofts eigene Site in der Tiefe, mit der das Ganze behandelt wird. Diese Site wurde zur ersten Adresse, bei der ein Java-Programmierer Informationen über ein Programm registriert, nachdem dieses fertiggestellt ist. Die Mitglieder der Gamelan-Mannschaft aktualisieren die Site täglich. Gamelan macht auf die besten Einsendungen auf der folgenden Seite aufmerksam:

<http://www.gamelan.com/special/picks.html>

Der Java Applet Rating Service

Um ein anderes Verzeichnis zu erreichen, das Java Applets bewertet, geben Sie in Ihrem Webbrowser folgende Adresse ein:

<http://www.jars.com>

Das Apfel-Logo des Java Applet Rating Service (JARS) sieht man in vielen Java-Applets, die auf Webseiten angeboten werden. Die JARS-Site wurde vor kurzem erweitert und bietet nun auch Neuigkeiten über die Sprache und damit verbundene Entwicklungen, Besprechungen von Entwicklungswerkzeugen und andere nützliche Informationen.

Das JavaWorld-Magazin

Eines der besten Magazine, das ins Leben gerufen wurde, um der Gemeinde der Java- Programmierer zu dienen, ist gleichzeitig auch das günstigste. JavaWorld ist kostenlos im World Wide Web unter der folgenden Adresse verfügbar:

<http://www.javaworld.com>

JavaWorld veröffentlicht häufig Artikel mit Kursen, Neuigkeiten zur Entwicklung mit Java und andere Beiträge, die monatlich aktualisiert werden. Die Tatsache, daß dieses Magazin nur im Web verfügbar ist, bietet einen Vorteil gegenüber einigen der gedruckten Wettbewerber (wie z.B. Java Report) im Bereich der How-To-Artikel. Wenn ein Artikel ein bestimmtes Programmierkonzept oder eine bestimmte Art der Programmierung beschreibt, kann JavaWorld ein Java-Applet anbieten, das dies demonstriert.

Java FAQs

Als Gegenstück zu den Java-FAQ-Listen (FAQ = Frequently Asked Questions = häufig gestellte Fragen), die auf der Website von JavaSoft verfügbar sind, haben Java-Programmierer, die Diskussionsgruppen im Internet nutzen, zusammen an deren eigenen Fragen- und Antwortenlisten gearbeitet.

Elliotte Rusty Harold, einer der Herausgeber der oben beschriebenen Website zu Java- Büchern, bietet auch eine aktuelle FAQ-Liste unter der folgenden Adresse an:

<http://sunsite.unc.edu/javafaq/javafaq.html>

Eine andere, ähnliche Ressource (»Unofficial Obscure Java FAQ« genannt) startete ursprünglich, um weniger häufig gestellte Fragen zu beantworten. Sie finden diesen FAQ auf der folgenden Site:

<http://k2.scl.cwru.edu/~gaunt/java/java-faq.html>

Java Newsgroups

Eine der besten Ressourcen sowohl für Neulinge als auch für erfahrene Java-Programmierer ist das Usenet. Ein internationales Netzwerk von Diskussionsgruppen, das den meisten Internet-Anwendern zur Verfügung steht. Im folgenden finden Sie Beschreibungen einiger der vielen Java-Diskussionsgruppen, die im Usenet verfügbar sind:

- [comp.lang.java.misc](#) - Obwohl diese Gruppe als Java-Diskussionsbereich für alle Themen dient, die nicht in eine der anderen Gruppen gehören, wird diese stärker verwendet als alle anderen. Mitte 1996 ersetzte diese Gruppe die Diskussionsgruppe comp.lang.java. Hier kann jedes Java-bezogene Thema diskutiert werden.
- [comp.lang.java.advocacy](#) - Diese Gruppe ist allen Diskussionen gewidmet, die wahrscheinlich hitzige oder vergleichende Debatten entstehen lassen. Wenn Sie die Vorzüge von Java gegenüber anderen Sprachen erörtern wollen, ist dies der richtige Ort dafür. Es kann sich lohnen, diese Gruppe zu konsultieren, wenn Sie wissen wollen, ob Java für ein Projekt, an dem Sie arbeiten, die richtige Wahl ist.
- [comp.lang.java.announce](#) - Diese Gruppe veröffentlicht Ankündigungen und Pressemitteilungen, die für die Java-Gemeinde wichtig sind. Die Gruppe wird moderiert, so daß alle Einsendungen geprüft werden, bevor sie an die Gruppe geschickt werden.
- [comp.lang.java.api](#) - In dieser Gruppe wird das API (Application Programming Interface) der Sprache Java, sprich die gesamte Bibliothek der Klassen, die mit dem JDK ausgeliefert wird, und andere Bibliotheken, die nicht direkt zum JDK gehören, diskutiert.
- [comp.lang.java.programmer](#) - Diese Gruppe beinhaltet Fragen und Antworten zur Programmierung mit Java, was diese Gruppe für Java-Programmierneulinge zu einer guten Quelle macht.
- [comp.lang.java.security](#) - Diese Diskussionsgruppe ist den Sicherheitsthemen rund um Java gewidmet, insbesondere in bezug auf die Ausführung von Java-Programmen und anderen ausführbaren Inhalten über das World Wide Web.
- [comp.lang.java.setup](#) - Diese Gruppe bietet eine Anlaufstelle, um Probleme bei der Installation von Java-Programmierwerkzeugen und ähnliche Themen zu diskutieren.
- [comp.lang.java.tech](#) - Die anspruchsvollste der Java-Diskussionsgruppen. Diese Gruppe ist der

Diskussion um die Implementierung der Sprache, der Portierung auf andere Maschinen, den Einzelheiten der Java Virtual Machine und ähnlichen Themen gewidmet.

Berufschancen

Wenn Sie Java lernen, um einen Job zu finden, sollten Sie sich einige der Java-bezogenen Job-Angebote im Web ansehen. Viele Ressourcen, die in diesem Anhang aufgeführt sind, verfügen über einen Bereich, der den Job-Angeboten gewidmet ist.

Wenn Sie daran interessiert sind, direkt bei JavaSoft unterzukommen, dann besuchen Sie die folgende Webseite:

<http://www.javasoft.com/aboutJavaSoft/jobs/index.html>

JavaWorld bietet eine Seite mit Job-Angeboten, die manchmal Möglichkeiten für Java-Entwickler aufführt:

<http://www.javaworld.com/javaworld/common/jw-jobop.html>

Um Arbeitgeber, die Mitarbeiter mit Java-Kenntnissen suchen, auf Ihre Fähigkeiten aufmerksam zu machen, können Sie sich selbst als Ressource für das Verzeichnis von Gamelan registrieren. Gamelan nimmt Sie dann in seine Site auf, und daraus könnten sich dann E-Mails mit Job-Angeboten ergeben. Um zu erfahren, wie die Registrierung bei Gamelan funktioniert, gehen Sie zu der folgenden Adresse im Bereich Add a Resource von Gamelan:

http://www.gamelan.com/submit/submit_person.shtml

Obwohl die folgende Webseite nicht direkt eine Ressource für Job-Angebote ist, ermöglicht es Ihnen die Website Career Path, die Stellenanzeigen von mehr als zwei Dutzend US-Zeitungen zu durchsuchen. Dieser Dienst ist kostenlos, Sie müssen sich allerdings registrieren. Sie finden dort mehr als 100.000 Stellenanzeigen, die Sie nach den Schlüsselwörtern Java oder Internet durchsuchen können. Besuchen Sie dazu die folgende Adresse:

<http://www.careerpath.com>

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Tag D

Die Konfiguration des Java Development Kit

Das Java Development Kit besteht aus einer Reihe von kommandozeilenorientierten Tools, die zur Erstellung, Kompilierung und zur Ausführung von Java-Programmen verwendet werden. Für das Kit ist die Kommandozeile nötig, da Sie die einzelnen Programme des Kit über die Eingabe des jeweiligen Namens und der notwendigen Argumente starten.

Die Konfiguration unter Windows 95 und Windows NT

Windows-95- und Windows-NT-Anwender erreichen die Kommandozeile (auch Eingabeaufforderung bzw. MS-DOS-Prompt genannt), indem sie in der Taskleiste Start | Programme | MS-DOS-Eingabeaufforderung wählen.

Wenn Sie das Kommando MS-DOS-Eingabeaufforderung wählen, öffnet sich ein Fenster, in das Sie Kommandos eingeben und MS-DOS-Befehle verwenden können, um die Ordner des Dateisystems zu durchsuchen. Der Start-Ordner ist der Ordner \Windows auf Ihrer ersten Festplatte (normalerweise C:\Windows).

Ein Beispiel für ein Kommando, das Sie eingeben können, sobald Sie das JDK installiert haben, wäre:

```
java -version
```

Dieses Kommando startet java.exe, den Java-Interpreter, der Teil des JDK ist. Das Argument -version veranlaßt den Interpreter dazu, seine Versionsnummer auszugeben.

Probieren Sie dieses Kommando einmal selbst aus. Wenn Sie das JDK 1.2 verwenden, sollte die folgende Nachricht als Reaktion ausgegeben werden:

```
java version "1.2"
```

Spätere Versionen des JDK 1.2 werden mit einer geringfügig anderen Versionsnummer antworten.

Wenn Sie einen Fehler erhalten, wie z.B. *falsche Versionsnummer* oder *Datei- oder Kommando nicht gefunden*, dann hat Ihr System Probleme, den Java-Interpreter zu finden. Um dies zu korrigieren, müssen Sie einige Korrekturen an der Konfiguration Ihres Systems vornehmen.

Bei der Behebung von Problemen bei der Konfiguration des JDK unter Windows sollten Sie als erstes herausfinden, wo sich java.exe auf Ihrem System befindet. Verwenden Sie den Dialog zur Suche nach Dateien (Start | Suchen | Dateien/Ordner in der Taskleiste), um nach der Datei java.exe auf der Festplatte, auf der Sie das JDK installiert haben, zu suchen.

Wenn Sie java.exe nicht finden, müssen Sie das JDK neu installieren und sicherstellen, daß die Installation die korrekte Größe aufweist, bevor Sie diese erneut ausführen.

Wenn Sie java.exe finden, kann es sein, daß Sie mehr als eine Version der Datei finden, falls sich

frühere Versionen des JDK auf Ihrem System befinden. Sehen Sie sich in der Spalte In Ordner des Such-Dialoges die Namen der Ordner an, die eine Version von java.exe enthalten. Einer von diesen sollte eine Referenz auf den Ordner enthalten, in dem Sie das JDK 1.2 installiert haben.

In Abbildung D.1 sind zwei Versionen von java.exe im Suchen-Dialog aufgeführt. Die eine, die sich in diesem Beispiel auf die aktuelle JDK-Installation bezieht, ist C:\jdk1.2\bin.

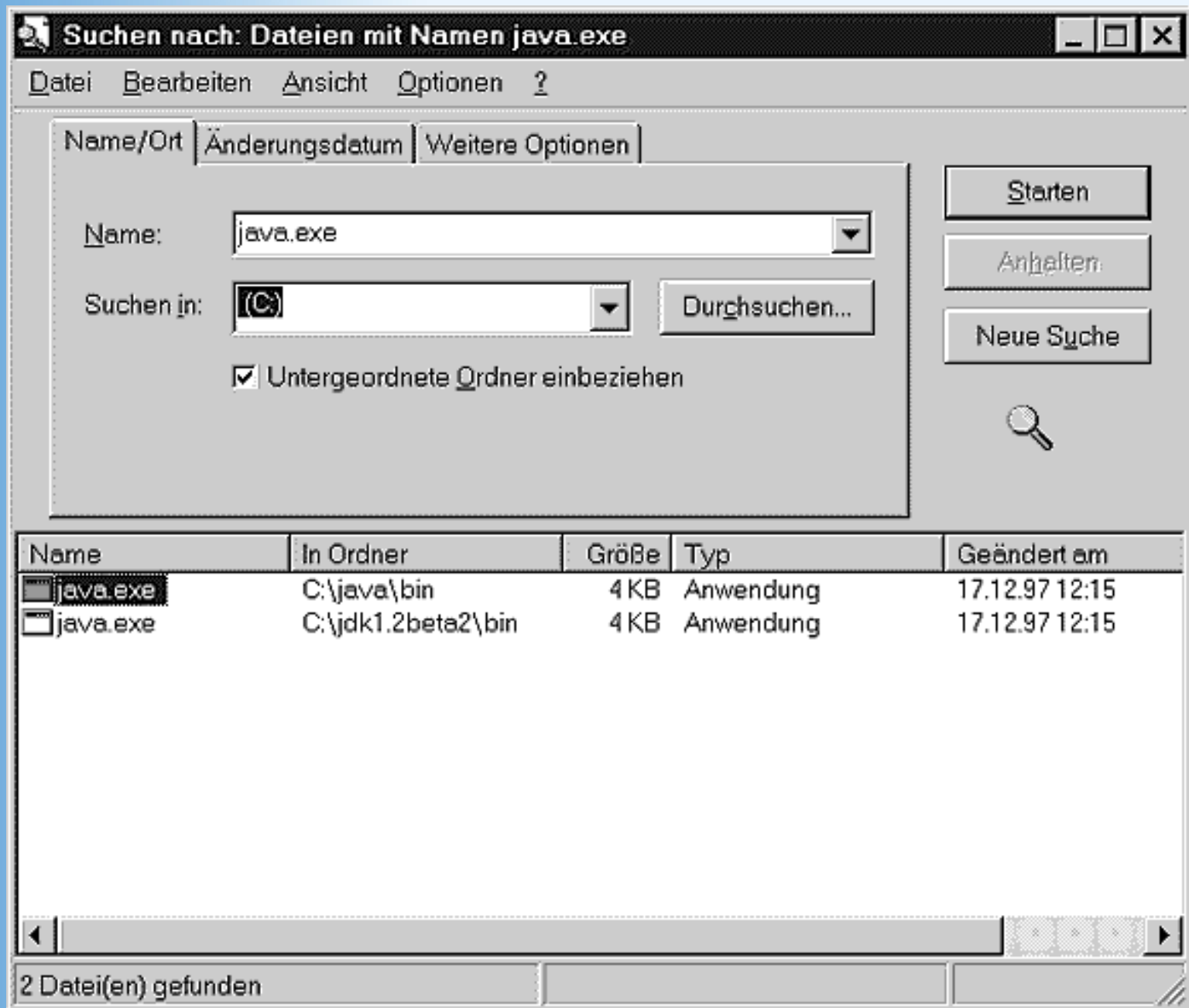


Abbildung D.1:
Das Ergebnis einer Dateisuche nach java.exe

Notieren Sie sich dieses Verzeichnis genau so, wie es in der Spalte In Ordner aufgelistet ist. Dies ist der Ordner des JDK-Pfades, und diesen werden Sie in Kürze verwenden.

Die PATH-Anweisung überprüfen

Um sicherzugehen, daß Ihr System am richtigen Ort nach den Dateien des JDK 1.2 sucht, müssen Sie sich die PATH-Einstellungen Ihres Systems ansehen. Die PATH-Anweisung legt die Pfade fest, in denen das Betriebssystem nach Programmen sucht, wenn man den Namen eines Programms an der

MS-DOS-Eingabeaufforderung eingibt (wie z.B. `java -version`).

Um die aktuellen Einstellungen von PATH anzeigen zu lassen, geben Sie das folgende Kommando an der MS-DOS-Eingabeaufforderung ein:

Path

Es wird eine Liste aller Ordner angezeigt, in denen Windows nach MS-DOS-Programmen sucht. Die einzelnen Einträge der Liste werden durch Strichpunkte voneinander getrennt. Hier ein Beispiel:

`PATH=C:\WINDOWS;C:\WINDOWS\COMMAND;C:\JDK1.2\BIN`

In dieser PATH-Liste bezieht sich der Eintrag `C:\JDK1.2\BIN` auf den Ordner des JDK im Suchpfad. In diesem Ordner sucht das System nach der Datei `java.exe`. Neben diesem Ordner sind noch zwei weitere Ordner in PATH gelistet - `C:\WINDOWS` und `C:\WINDOWS\COMMAND`.

Ihre PATH-Einstellungen sollten eine Referenz auf den Ordner des JDK enthalten. (Die Groß-/Kleinschreibung spielt in diesem Fall keine Rolle - `C:\JDK1.2\BIN` wird dabei genauso interpretiert wie `C:\jdk1.2\bin`.)

Wenn der JDK-Ordner nicht in den PATH-Einstellungen enthalten ist, müssen Sie Ihre PATH-Einstellung bearbeiten und den JDK-Ordner hinzufügen.

Die PATH-Einstellungen ändern

Die PATH-Einstellungen ändern Sie, indem Sie die `autoexec.bat` editieren. Dies ist eine Text-Datei, die sich im Stammordner der ersten Festplatte Ihres Systems befindet (normalerweise ist dies `C:`).

Um die Einstellungen auf einem Windows-NT-System zu ändern, müssen Sie die Datei zur Bearbeitung nicht öffnen. Statt dessen öffnen Sie über die Taskleiste die Systemsteuerung (Start | Einstellungen | Systemsteuerung).

Um die PATH-Einstellungen auf einem Windows-95-System zu ändern, müssen Sie die Datei `autoexec.bat` bearbeiten. Klicken Sie dazu mit der rechten Maustaste auf die Datei, und wählen Sie das Kommando Bearbeiten aus dem Kontextmenü, um die Datei in den Editor von Windows 95 zu laden.

In der Datei `autoexec.bat` werden Sie eine Menge Dinge finden, die sehr technisch aussehen und für den MS-DOS-Neuling unverständlich sein werden.

Suchen Sie nach einer Zeile, die mit dem Text `PATH=`, `SET PATH=` oder `PATH` gefolgt von einem Leerzeichen beginnt.

Problem: keine PATH-Anweisung

Wenn Sie keine PATH-Anweisung finden, sollten Sie eine in die Datei `autoexec.bat` einfügen. Gehen Sie dazu an das Ende der Datei, und fügen Sie eine Leerzeile ein. Geben Sie in dieser Zeile den `PATH=` gefolgt von dem Ordernamen des JDK. Die folgende Zeile können Sie verwenden, wenn sich Ihr JDK im Ordner `C:\jdk1.2` auf Ihrem System befindet:

`PATH=C:\jdk1.2\bin`

Speichern Sie die Datei, nachdem Sie diese Änderung vorgenommen haben. Sie müssen Ihr System neu starten, bevor diese Änderungen Wirkung zeigen. Wenn sich die Datei `java.exe` in dem Ordner befindet, den Sie in der PATH-Anweisung angegeben haben, sollten Sie in der Lage sein, `java -version` erfolgreich auszuführen.

Problem: der Ordner des JDK 1.2 ist nicht in der PATH-Anweisung

Wenn Sie eine PATH-Anweisung in der autoexec.bat-Datei finden, die keine Referenz auf den Ordner des JDK 1.2 beinhaltet, dann suchen Sie nach dem Text %JAVA_HOME% in dieser Zeile gefolgt von einem Text wie \bin.

Falls Sie %JAVA_HOME% finden sollten, löschen Sie es und den Text, der darauf folgt, bis zum nächsten Strichpunkt (allerdings ohne den Strichpunkt zu löschen). Ersetzen Sie es durch den korrekten Pfad des JDK 1.2, den Sie sich zuvor aufgeschrieben haben. Stellen Sie sicher, daß Sie keine Strichpunkte, die zur Abtrennung der einzelnen Ordner voneinander verwendet werden, löschen.

Falls Sie %JAVA_HOME% in der PATH-Anweisung nicht finden, platzieren Sie Ihren Cursor am Ende der Zeile mit der PATH-Anweisung. Geben Sie an dieser Stelle ein Semikolon gefolgt von dem Namen des Pfades des JDK ein. Wenn C:\jdk1.2\bin der Pfad Ihres JDK ist, dann sollte das Ende Ihrer PATH-Anweisung wie das Ende der folgenden Zeile aussehen:

```
PATH=C:\WINDOWS;C:\WINDOWS\COMMAND;C:\jdk12\bin
```

In der PATH-Anweisung sollte auf keine andere Version des JDK verwiesen werden. Wenn Sie einen Pfad entdecken, der sich auf eine frühere Version bezieht, löschen Sie die Referenz auf diesen Ordner. Falls sich daraus eine Zeile, die zwei Strichpunkte nacheinander (;;) enthält, ergeben sollte, löschen Sie einen der Strichpunkte.

Speichern Sie diese Datei, nachdem Sie diese Änderungen gemacht haben. Sie müssen Ihr System neu starten, bevor die PATH-Anweisung Wirkung zeigt. Wenn sich die Datei java.exe in dem Ordner befindet, den Sie in der PATH-Anweisung angegeben haben, sollten Sie in der Lage sein, java -version erfolgreich auszuführen.

Class-Not-Found-Fehler beheben

Java-Programme werden mit dem JDK kompiliert, indem Sie den Java-Compiler javac mit dem Namen der Quelldatei als Argument aufrufen. Wenn Sie sich z.B. in demselben Ordner wie die Quelldatei HalloDan.java befinden würden, könnten Sie diese mit dem folgenden Kommando kompilieren:

```
javac HalloDan.java
```



Falls Sie bisher noch nicht versucht haben eine Datei zu kompilieren, um das JDK zu testen, können Sie dies mit der Datei HalloDan.java, die Sie auf der CD zum Buch finden, tun.

Wenn Sie beim Versuch, eine Quelldatei mit dem Java-Compiler zu kompilieren, eine Fehlermeldung Class not found erhalten, dann sollten Sie als erstes überprüfen, ob Sie den Namen unter Beachtung der Groß-Klein-Schreibung richtig geschrieben haben.

Stellen Sie sicher, daß Sie sich in dem Ordner befinden, der die Quelldatei enthält, die Sie zu kompilieren versuchen. Außerdem sollten Sie den Dateinamen doppelt überprüfen.

Windows-Anwender können das DIR-Kommando an der MS-DOS-Eingabeaufforderung verwenden, um alle Dateien in einem Ordner auflisten zu lassen. Der vollständige Dateiname der Datei wird in der Spalte, die sich am weitesten rechts befindet, angezeigt (siehe Abbildung D.2). Die abgekürzten

Dateinamen an der linken Seite der Liste sollten Sie ignorieren - diese werden von Windows intern zur Verwaltung der Dateien verwendet.

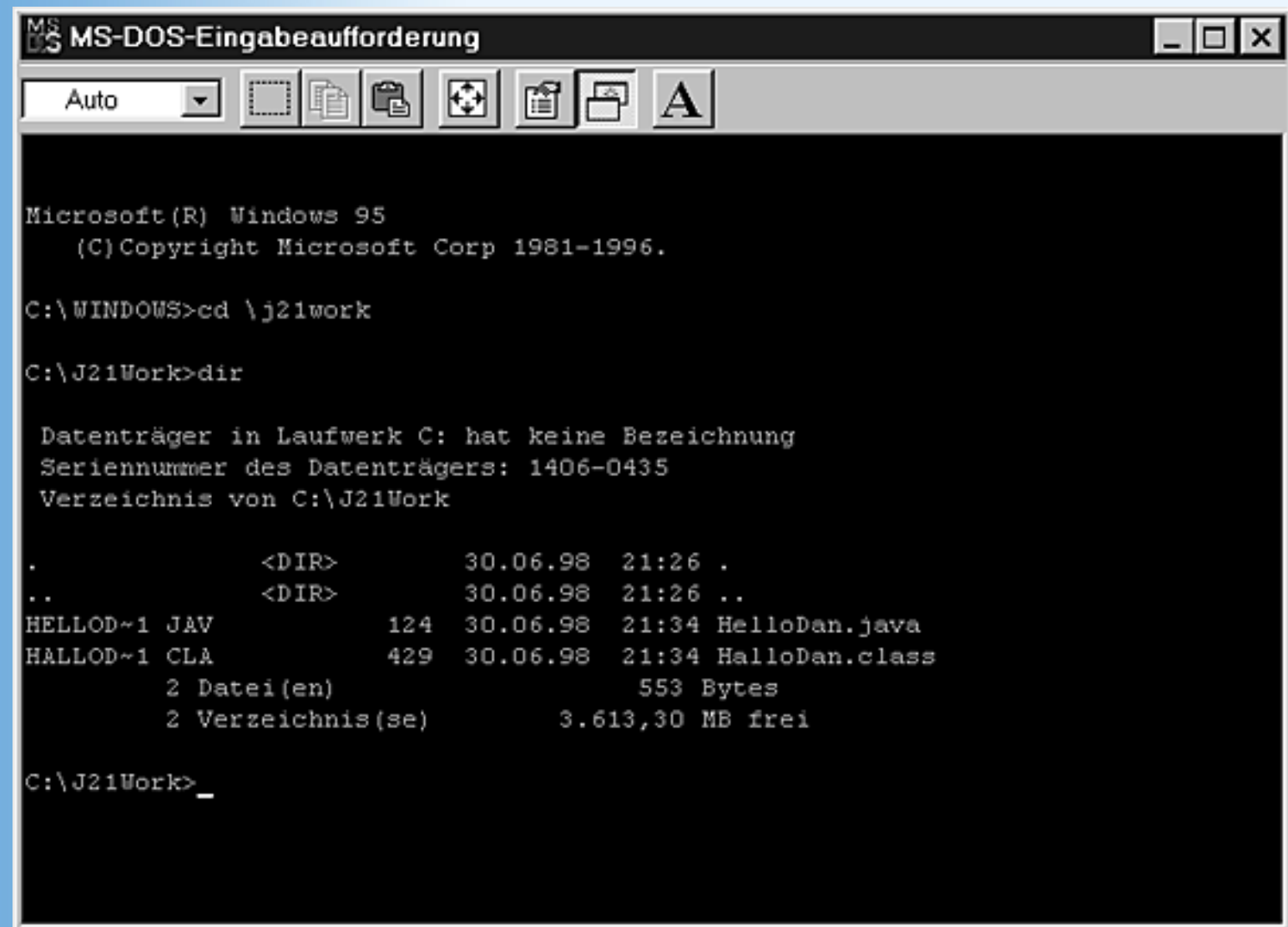


Abbildung D.2:
Eine Dateiliste in einem MS-DOS-Fenster.

Wenn der Name der Quelldatei korrekt ist und es keine Fehler in dem Java-Quellcode selbst gibt, dann hat Ihr System Probleme damit, die Datei `tools.jar` zu finden. Diese Datei beinhaltet alle Java-Klassen, die zur erfolgreichen Kompilierung und Ausführung von Java-1.2-Programmen benötigt werden.

Das JDK sucht nach der Datei `tools.jar` auf zwei Arten. Zunächst verwendet es die `CLASSPATH`-Einstellung des Systems (sofern eine gesetzt wurde). Als nächstes sucht es nach der Datei `java.exe` und verwendet den Speicherort dieser Datei, um herauszufinden, wo sich `tools.jar` befindet.

Die meisten `Class-not-found`-Fehler können behoben werden, indem man die `CLASSPATH`-Einstellung verwendet, um den Speicherort von `tools.jar` anzugeben.

In Windows 95 oder Windows NT verwenden Sie die Dateisuche (Start | Suchen | Dateien/Ordner in der Taskleiste), um nach der Datei `tools.jar` auf demselben Laufwerk zu suchen, auf dem Sie auch das JDK installiert haben (siehe auch Abbildung D.3).

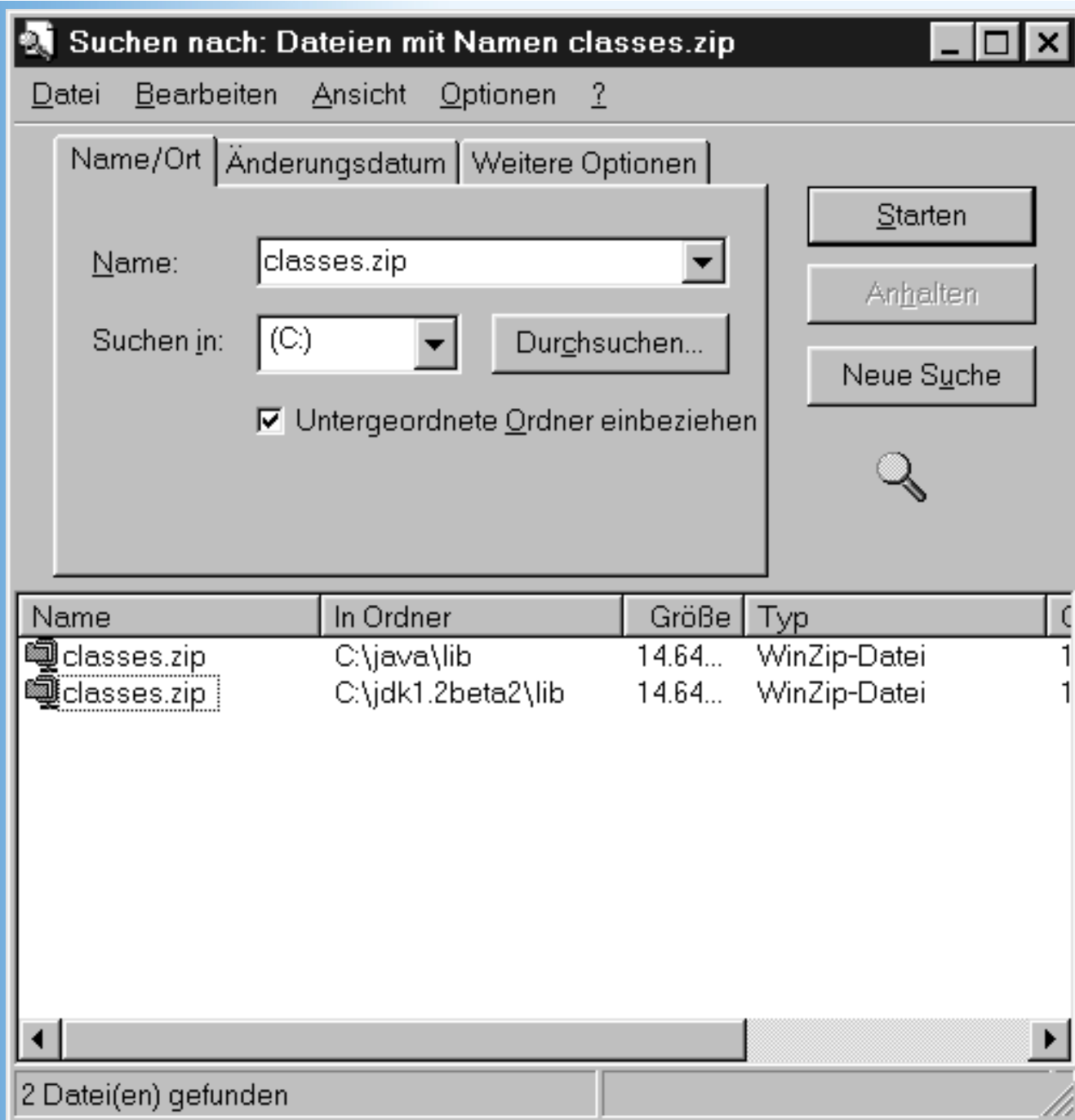


Abbildung D.3:
Das Ergebnis einer Suche mit dem Suchen-Dialog nach der Dateitools.jar.

Manchmal wird sich mehr als eine Datei mit dem Namen tools.jar auf Ihrem System befinden. Einige davon werden von früheren Versionen des JDK und anderen Java- Entwicklungstools stammen. Andere wiederum könnten von Webbrowsern verwendet werden, die Java unterstützen.

Sehen Sie sich in der Spalte In Ordner den vollen Namen (inklusive des Laufwerkskennbuchstabens) eines jeden Ordners an, der eine Datei mit dem Namen tools.jar enthält. Versuchen Sie den Eintrag zu finden, der sich auf den Ordner bezieht, in dem Sie das JDK 1.2 installiert haben (in Abbildung D.3 ist dies C:\jdk1.2\lib).

Schreiben Sie sich den gesamten Pfadnamen dieses Ordners auf. Dieser Pfadname gefolgt von \tools.jar sollte Ihre CLASSPATH-Einstellung sein.

Die CLASSPATH-Anweisung prüfen

Stellen Sie sicher, daß Ihr System an der richtigen Stelle nach der JDK-1.2-Version von tools.jar sucht. Dazu prüfen Sie die CLASSPATH-Einstellung Ihres Systems.

Windows-Anwender können die aktuellen CLASSPATH-Einstellungen überprüfen, indem sie folgendes an der MS-DOS-Eingabeaufforderung eingeben:

```
echo %CLASSPATH%
```

Stellen Sie sicher, daß Sie die Prozentzeichen (%) vor und nach dem Ausdruck CLASSPATH angeben. Wenn in Ihrem System ein CLASSPATH festgelegt ist, sehen Sie eine Liste aller Ordner und Dateien, in denen die JDK-Tools nach Java-Klassen suchen. Die einzelnen Ordner in der Liste sind durch Strichpunkt voneinander getrennt. Im folgenden ein Beispiel:

```
.;C:\jdk1.2\lib\tools.jar
```

In dieser CLASSPATH-Liste bezeichnet der Eintrag C:\jdk1.2\lib\tools.jar einen Ort, von dem Java-Klassen geladen werden. Zusätzlich befindet sich in der Auflistung ein Punkt (.) als erster Eintrag. Dieser Eintrag stellt sicher, daß JDK-Utilities auch in dem aktuellen Ordner nach Klassen suchen, die anderswo nicht gefunden werden.

Falls CLASSPATH keine Referenz auf tools.jar des JDK 1.2 beinhaltet, müssen Sie die CLASSPATH-Einstellungen bearbeiten und dies entsprechend einfügen.

Die CLASSPATH-Einstellungen ändern

Die CLASSPATH-Einstellungen ändern Sie, indem Sie die autoexec.bat editieren. Dies ist eine Text-Datei, die sich im Stammordner der ersten Festplatte Ihres Systems befindet (normalerweise ist dies C:).

Um die Einstellungen auf einem Windows-NT-System zu ändern, müssen Sie die Datei zur Bearbeitung nicht öffnen. Statt dessen öffnen Sie über die Taskleiste die Systemsteuerung (Start | Einstellungen | Systemsteuerung).

Um die CLASSPATH-Einstellungen auf einem Windows-95-System zu ändern, müssen Sie die Datei autoexec.bat bearbeiten. Klicken Sie dazu mit der rechten Maustaste auf die Datei, und wählen Sie das Kommando Bearbeiten aus dem Kontextmenü, um die Datei in den Editor von Windows 95 zu laden.

In der Datei autoexec.bat werden Sie eine Menge Dinge finden, die sehr technisch aussehen und für den MS-DOS-Neuling unverständlich sein werden.

Suchen Sie nach einer Zeile, die mit dem Text CLASSPATH=, SET CLASSPATH= oder CLASSPATH gefolgt von einem Leerzeichen beginnt.

Problem: keine CLASSPATH-Anweisung

Wenn Sie keine CLASSPATH-Anweisung finden, sollten Sie eine in die Datei autoexec.bat einfügen. Gehen Sie dazu an das Ende der Datei, und fügen Sie eine Leerzeile ein. Geben Sie in dieser Zeile den CLASSPATH= gefolgt von dem Ordernamen des JDK. Die folgende Zeile können Sie verwenden, wenn sich Ihr JDK im Ordner C:\jdk1.2 auf Ihrem System befindet:

```
CLASSPATH=.;C:\jdk1.2\lib\tools.jar
```

Speichern Sie die Datei, nachdem Sie diese Änderung vorgenommen haben. Sie müssen Ihr System neu starten, bevor diese Änderungen Wirkung zeigen. Wenn sich die Datei tools.jar in dem Ordner befindet, den Sie in der CLASSPATH-Anweisung angegeben haben, sollten Sie in der Lage sein,

Programme erfolgreich zu kompilieren.

Problem: der Ordner des JDK 1.2 ist nicht in der CLASSPATH-Anweisung

Wenn Sie eine CLASSPATH-Anweisung in der autoexec.bat-Datei finden, die keine Referenz auf den korrekten Pfad von tools.jar beinhaltet, dann suchen Sie nach dem Text %JAVA_HOME% in dieser Zeile.

Sie finden eventuell %JAVA_HOME% gefolgt von dem Text wie \bin\tools.jar, wie z.B. in CLASSPATH=%JAVA_HOME%\lib\tools.jar oder CLASSPATH=%JAVA_HOME%\..\lib\tools.jar.

Falls Sie %JAVA_HOME% finden sollten, löschen Sie es und den Text, der darauf folgt, bis zum nächsten Strichpunkt (allerdings ohne den Strichpunkt zu löschen). Ersetzen Sie es durch den korrekten Pfad zu tools.jar. Stellen Sie sicher, daß Sie keine Strichpunkte, die zur Abtrennung der einzelnen Ordner voneinander verwendet werden, löschen.

Falls Sie %JAVA_HOME% in der CLASSPATH-Anweisung nicht finden, plazieren Sie Ihren Cursor am Ende der Zeile mit der CLASSPATH-Anweisung. Geben Sie an dieser Stelle ; gefolgt von dem korrekten Pfad zu tools.jar ein. Wenn C:\jdk1.2\lib\tools.jar der Pfad zu tools.jar ist, sollte das Ende Ihrer CLASSPATH-Anweisung wie das Ende der folgenden Zeile aussehen:

```
CLASSPATH= . ; C : \DEV\CHATSERVER\ ; C : \jdk12\lib\tools.jar
```

In der CLASSPATH-Anweisung sollte auf keine andere Version des JDK verwiesen werden. Wenn Sie einen Pfad entdecken, der sich auf eine frühere Version bezieht, dann löschen Sie die Referenz auf diesen Ordner. Falls sich daraus eine Zeile, die zwei Strichpunkte nacheinander (;;) enthält, ergeben sollte, löschen Sie einen der Strichpunkte.

Speichern Sie diese Datei, nachdem Sie diese Änderungen gemacht haben. Sie müssen Ihr System neu starten, bevor die CLASSPATH-Anweisung Wirkung zeigt. Wenn sich die Datei tools.jar in dem Ordner befindet, den Sie in der CLASSPATH-Anweisung angegeben haben, sollten Sie in der Lage sein, Programme, wie z.B. HalloDan erfolgreich zu kompilieren und auszuführen.

Die Konfiguration unter Unix

Um das JDK auf einem Solaris-System zu konfigurieren, fügen Sie das Verzeichnis java/bin oder jdk/bin in den Ausführrpfad ein. Dies erreichen Sie normalerweise dadurch, daß Sie einen Eintrag wie den folgenden in Ihre .profile, .cshrc oder .login- Datei einfügen:

```
set path= (~ / java / bin / $path)
```

Diese Zeile geht davon aus, daß Sie das JDK im Verzeichnis Java Ihres Home-Verzeichnisses installiert haben. Sollte sich das JDK an anderer Stelle befinden, muß dies entsprechend hier berücksichtigt werden.

Diese Änderungen werden erst dann wirksam, wenn Sie sich ausgeloggt und wieder eingeloggt haben, oder Sie verwenden das Kommando source zusammen mit dem Namen der Datei, die Sie geändert haben. Wenn Sie die Datei .login geändert haben, sieht das source-Kommando wie folgt aus:

```
source ~/.login
```


Class-Not-Found-Fehler beheben

Um Class-not-found-Fehler auf Solaris-Systemen zu beheben, sollten Sie sicherstellen, daß die CLASSPATH-Umgebungsvariable beim Login nicht automatisch gesetzt wird.

Um zu überprüfen, ob CLASSPATH gesetzt ist, geben Sie folgendes an der Eingabeaufforderung ein:
`echo $CLASSPATH`

Falls CLASSPATH gesetzt wurde, können Sie diese Einstellung über das folgende Kommando aufheben:

```
unsetenv CLASSPATH
```

Um diese Änderung permanent zu machen, sollten Sie das Kommando, das CLASSPATH setzt, aus Ihrer .profile, .cshrc oder .login entfernen.

Diese Änderungen werden erst dann wirksam, wenn Sie sich ausgeloggt und wieder eingeloggt haben, oder Sie verwenden das Kommando source zusammen mit dem Namen der Datei, die Sie geändert haben. Wenn Sie die Datei .login geändert haben, sieht das source-Kommando wie folgt aus:

```
source ~/.login
```

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).

Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Tag E

Texteditoren und das JDK

Im Gegensatz zu Java-Entwicklungswerkzeugen wie Visual Café und Java Workshop von SunSoft besitzt das Java Development Kit keinen Texteditor für die Erstellung der Quelldateien.

In diesem Anhang lernen Sie, wie Sie einen Editor für die Arbeit mit dem JDK auswählen und wie Sie Ihr System dafür entsprechend konfigurieren.

Die Auswahl eines Texteditors

Damit ein Editor oder ein Textverarbeitungsprogramm mit dem JDK zusammenarbeiten kann, muß es in der Lage sein, Textdateien ohne jegliche Formatierungen zu speichern.

Dieses Feature hat in den verschiedenen Editoren unterschiedliche Namen. Suchen Sie beim Speichern oder beim Festlegen der Dokumenteigenschaften nach einer Format-Option, die so ähnlich heißt wie eine aus der folgenden Liste:

- Text
- ASCII-Text
- DOS-Text
- Nur-Text

Wenn Sie Windows 95 verwenden, finden Sie einige Editoren, die mit dem Betriebssystem geliefert werden.

Der *Windows Editor* (auch Notepad; Start | Programme | Zubehör | Editor) ist ein Editor, der nur das Nötigste bietet, um Texte zu bearbeiten, und nur mit reinen Textdateien arbeitet. Außerdem kann er immer nur ein Dokument zur selben Zeit verarbeiten.

Windows WordPad (Start | Programme | Zubehör | WordPad) ist eine Stufe über dem Editor angesiedelt. Es kann mehr als eine Datei gleichzeitig verarbeiten und sowohl mit reinem Text als auch Microsoft-Word-Formaten umgehen. Zusätzlich »merkt« es sich die letzten paar Dokumente, mit denen es gearbeitet hat. Diese können Sie direkt aus dem Datei-Menü auswählen.

DOS Edit, das Sie über die Eingabeaufforderung über das Kommando edit aufrufen können, ist ein weiterer einfacher Editor, der Dokumente mit reinem Text verarbeitet. Einem Windows-95-Anwender, der nicht mit MS-DOS vertraut ist, wird dieser Editor ziemlich roh vorkommen, allerdings hat dieser Editor ein Feature, das man sowohl beim Windows-Editor als auch bei WordPad vergeblich sucht: Edit zeigt die Zeilennummer der Zeile, in der sich die Einfügemarke gerade befindet, an.

Die Fähigkeit, die Zeilennummer der Zeile, die Sie gerade bearbeiten, anzuzeigen, gibt es in DOS Edit und einigen anderen Texteditoren. Die Numerierung beginnt mit 1 bei der obersten Zeile und wird mit den Folgezeilen weitergeführt. Abbildung E.1 zeigt Edit; die Zeilennummer wird an der unteren rechten Ecke des Programmfensters angezeigt.

```

MS-DOS-Eingabeaufforderung - EDIT
Auto
Datei Bearbeiten Suchen Fenster Optionen Hilfe
C:\J21Work\range.java

class RangeClass {

    int[] makeRange(int lower, int upper) {

        int arr[] = new int[ (upper - lower) + 1 ];

        for (int i = 0; i < arr.length; i++) {
            arr[i] = lower++;
        }
        return arr;
    }

    public static void main(String arguments[]) {
        int theArray[];

        RangeClass theRange = new RangeClass();
        theArray = theRange.makeRange(1, 10);
        System.out.print("The array: [ ");
        for (int i = 0; i < theArray.length; i++) {
            System.out.print(theArray[i] + " ");
        }
        System.out.println("]");
    }
}

F1=Hilfe Zeile:4 Spalte:1

```

Abbildung E.1:
Eine Java-Quelldatei in DOS Edit.

Zeilennummern sind in Java-Programmen sehr hilfreich, da viele Java-Compiler die Zeilennummern angeben, in denen ein Fehler aufgetreten ist. Sehen Sie sich einmal die folgende Fehlermeldung an, die der Compiler des JDK erzeugt hat.

Palindrome.java:2: Class Font not found in type declaration.

In dieser Fehlermeldung befindet sich hinter dem ersten Doppelpunkt eine Nummer (hinter dem Namen der Quelldatei). Diese Nummer bezieht sich auf die Zeile, die den Fehler ausgelöst hat.

Mit einem Texteditor, der Zeilennummern anzeigt, können Sie direkt zu dieser Zeile springen und nach dem Fehler suchen. In einer kommerziellen Java-Programmierungsumgebung stehen meist bessere Möglichkeiten zur Fehlersuche zur Verfügung. Die Anwender des JDK müssen allerdings über die Zeilennummer nach Fehlern suchen, die der Compiler javac gemeldet hat. Aus diesem Grund ist es am sinnvollsten, einen Texteditor zu verwenden, der Zeilennummern anzeigt.

Dateitypen in Windows 95 registrieren

Nachdem Sie einen Texteditor ausgewählt haben, sollten Windows-95-Anwender ihren Editor mit den Dateierweiterung .java verknüpfen. Dadurch wird es möglich, eine .java-Quelldatei durch einen Doppelklick auf den Namen der Datei in einem Ordner zu öffnen. Außerdem hält es Editoren wie den Windows-Editor davon ab, fälschlicherweise die Erweiterung .txt an .java-Quelldateien anzuhängen.

Um eine Dateierweiterung zu registrieren, benötigen Sie als erstes eine Datei, mit der Sie arbeiten können. Öffnen Sie unter Windows 95 einen Ordner, und erzeugen Sie eine neue Textdatei, indem Sie

Datei | Neu | Textdatei aus der Menüleiste des Ordners wählen (siehe Abbildung E.2).

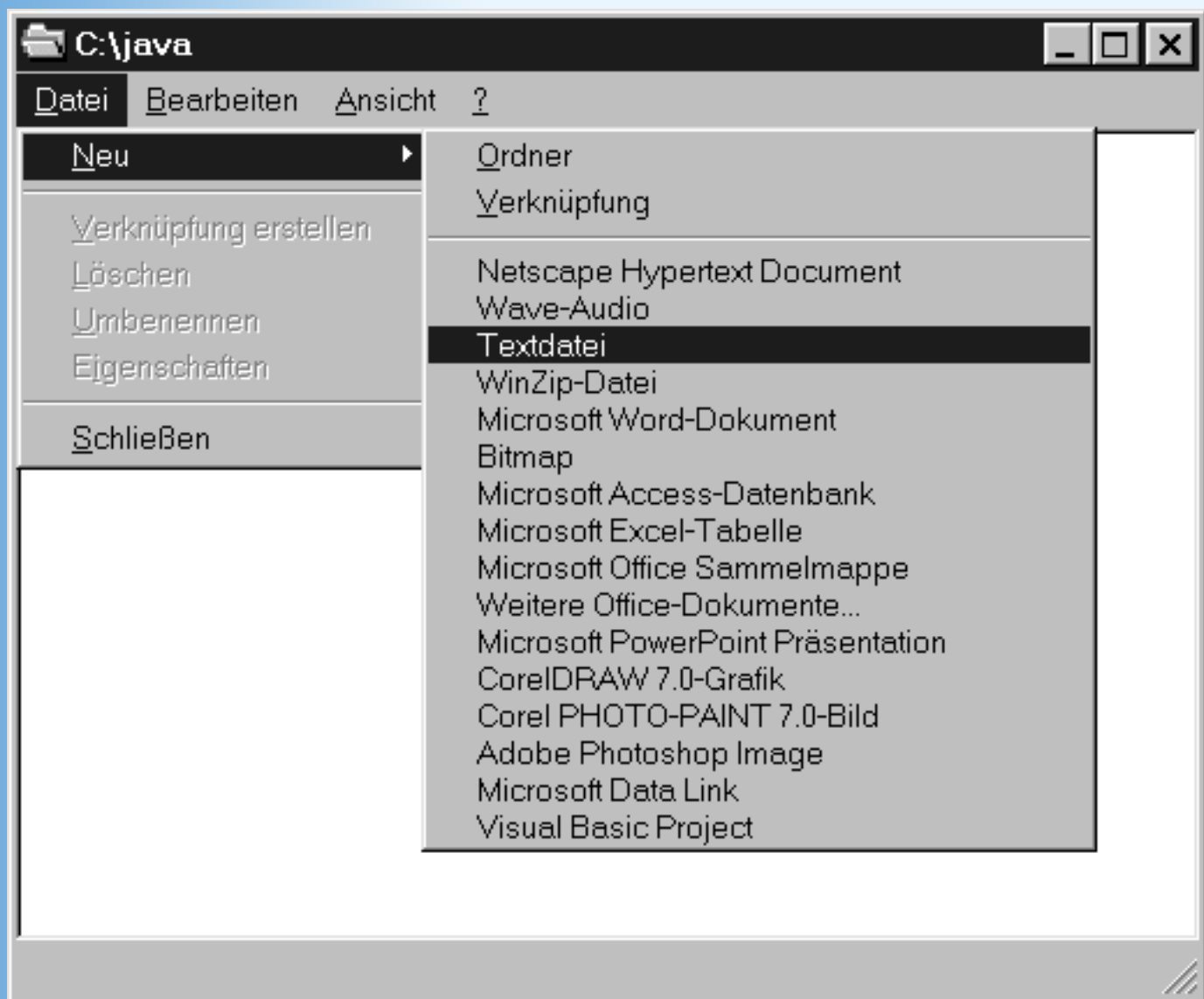


Abbildung E.2:
Eine neue Textdatei in einem Windows-95-Ordner erstellen.

Es wird eine neue Textdatei mit dem Namen Neufassung von Textdatei.txt erstellt. Sie können diese sofort umbenennen. Benennen Sie diese Datei in Irgendwie.java um. Klicken Sie nun doppelt auf diese Datei. Wenn dieser Dateityp in Ihrem System noch nicht registriert ist, erscheint der Dialog Öffnen mit. Hier können Sie in der Liste der Programme ein Programm wählen, mit dem diese Dateien in Zukunft geöffnet werden sollen. Wie Sie mit diesem Dialog umgehen, erfahren Sie im Abschnitt »Einen neuen Dateityp registrieren«.

Geschieht etwas anderes und dieser Dialog wird nicht angezeigt, dann müssen Sie die bestehende Verknüpfung mit der Erweiterung .java löschen, bevor Sie eine neue anlegen können.

Eine bestehende Verknüpfung mit einer Dateierweiterung löschen

Wenn in Ihrem System bereits irgend etwas mit der Dateierweiterung .java verknüpft ist, können Sie diese Verknüpfung aus einem beliebigen Windows-95-Ordner heraus entfernen. Wählen Sie Ansicht | Optionen aus der Menüleiste des Ordners. Es erscheint der Dialog Optionen. Dieser enthält drei Registerkarten. Holen Sie die Registerkarte Dateitypen in den Vordergrund (siehe auch Abbildung E.3).

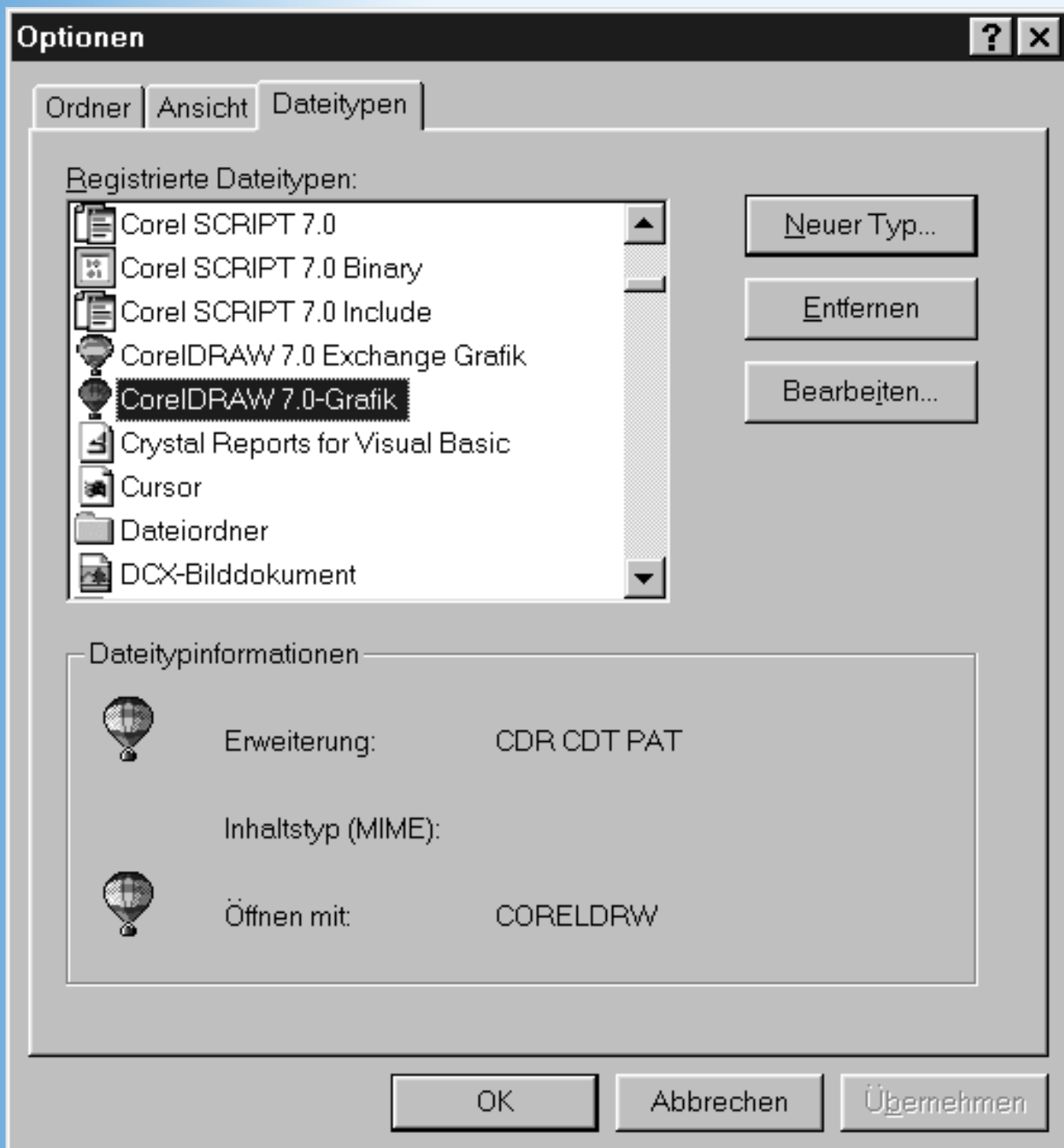


Abbildung E.3:
Die Registerkarte Dateitypen.

Im Listenfeld der registrierten Dateitypen sehen Sie alle Dateitypen, die mit Programmen auf Ihrem System verknüpft sind. Sobald Sie einen der Einträge in diesem Listenfeld markieren, werden in zwei anderen Feldern Informationen darüber angezeigt:

- Im Feld Erweiterung werden alle Dateinamenserweiterungen angezeigt, die diesem Dateityp zugeordnet sind.
- Im Feld Öffnen mit wird das Programm angezeigt, das verwendet wird, um diesen Dateityp zu öffnen.

Der Dateityp CorelDraw-7.0-Grafik in Abbildung E.4 besitzt z.B. drei Erweiterungen: CDR, CDT und PAT. Jede Datei mit einer dieser Erweiterungen kann mit dem Programm CORELDRW (das

Grafikprogramm CorelDraw) geöffnet werden.

Scrollen Sie durch die Liste der registrierten Dateitypen, bis Sie einen Typ finden, in dessen Erweiterungsfeld JAVA zu sehen ist. Am wahrscheinlichsten werden Sie bei einem Eintrag wie Java-Dateien oder Java-Programme fündig. Dies muß aber bei Ihrem System nicht der Fall sein.

Sobald Sie den richtigen Dateityp gefunden haben, müssen Sie die bestehende Verknüpfung löschen, um sie durch eine neue zu ersetzen. Markieren Sie dazu den entsprechenden Eintrag (falls das noch nicht der Fall ist), und klicken Sie auf die Schaltfläche Entfernen. Klicken Sie anschließend auf Ja, um diese Aktion zu bestätigen.

Nachdem Sie dies erledigt haben, können Sie eine neue Verknüpfung für die Erweiterung .java erstellen.

Einen neuen Dateityp registrieren

Wenn Sie auf eine Datei doppelt klicken, für deren Dateierweiterung es keine bekannte Assoziation gibt, wird der Dialog Öffnen mit angezeigt (siehe auch Abbildung E.4).



Abbildung E.4:
Eine Dateinamenserweiterung mit einem Programm verknüpfen.

Führen Sie die folgenden Schritte aus, um eine Verknüpfung für .java-Dateien zu erstellen:

- Geben Sie in dem Textfeld Beschreibung von '.java'-Dateien *Java-Quelldateien* oder etwas Ähnliches ein.
- Suchen Sie in dem Listenfeld Wählen Sie das gewünschte Programm den Texteditor oder das Textverarbeitungsprogramm, das Sie für Java-Quelldateien verwenden wollen. Falls Sie es nicht finden, klicken Sie auf die Schaltfläche Andere und suchen das Programm selbst. Wenn Sie DOS Edit verwenden wollen, finden Sie dies in dem Ordner \Windows\Command (auf den meisten Systemen - Windows ist dabei das Installationsverzeichnis von Windows). Der Name der Programmdatei ist edit oder edit.exe.
- Stellen Sie sicher, daß Sie das Kontrollkästchen Datei immer mit diesem Programm öffnen markieren.

Wenn Sie auf OK klicken, um diese Einstellungen zu bestätigen, wird die Datei Irgendwie.java und andere Dateien, die die Erweiterung .java haben, in dem gewählten Editor geöffnet.

Ein Symbol mit einem Dateityp verknüpfen

Nachdem Sie .java-Dateien mit Ihrem Editor der Wahl verknüpft haben, wird standardmäßig den .java-Dateien auf Ihrem System ein Symbol zugewiesen.

Wenn Sie statt dessen ein anderes Symbol verwenden wollen, wählen Sie in der Menüleiste eines Ordners den Eintrag Ansicht | Optionen und holen das Register Dateitypen in den Vordergrund. Scrollen Sie durch die Liste der registrierten Dateitypen, und suchen Sie nach dem Eintrag für die Dateierweiterung JAVA.

Klicken Sie auf Bearbeiten, um das Fenster Dateityp bearbeiten zu öffnen (siehe Abbildung E.5).

Klicken Sie auf Anderes Symbol. In dem darauffolgenden Dialog können Sie ein anderes Symbol auswählen, das für alle .java-Dateien angezeigt wird.

Wenn Sie eines dieser Symbole aus dem Fenster Aktuelles Symbol verwenden wollen, wählen Sie dieses aus und klicken auf OK, um die Änderung vorzunehmen.

Wenn Sie dagegen nach anderen Symbolen suchen wollen, klicken Sie auf die Schaltfläche Durchsuchen, um die Dateien auf Ihrem System zu durchsuchen und die Symbole anzusehen, die diese enthalten.

Sie können beliebige Dateien öffnen, Windows-95-Programme oder .DLL-Dateien, um nachzusehen, welche Symbole diese enthalten. Diese werden in dem Fenster Aktuelles Symbol angezeigt, nachdem Sie eine Datei ausgewählt haben.

Sobald Sie ein Symbol finden, das Ihnen gefällt, markieren Sie es und klicken auf OK, um es zu wählen.

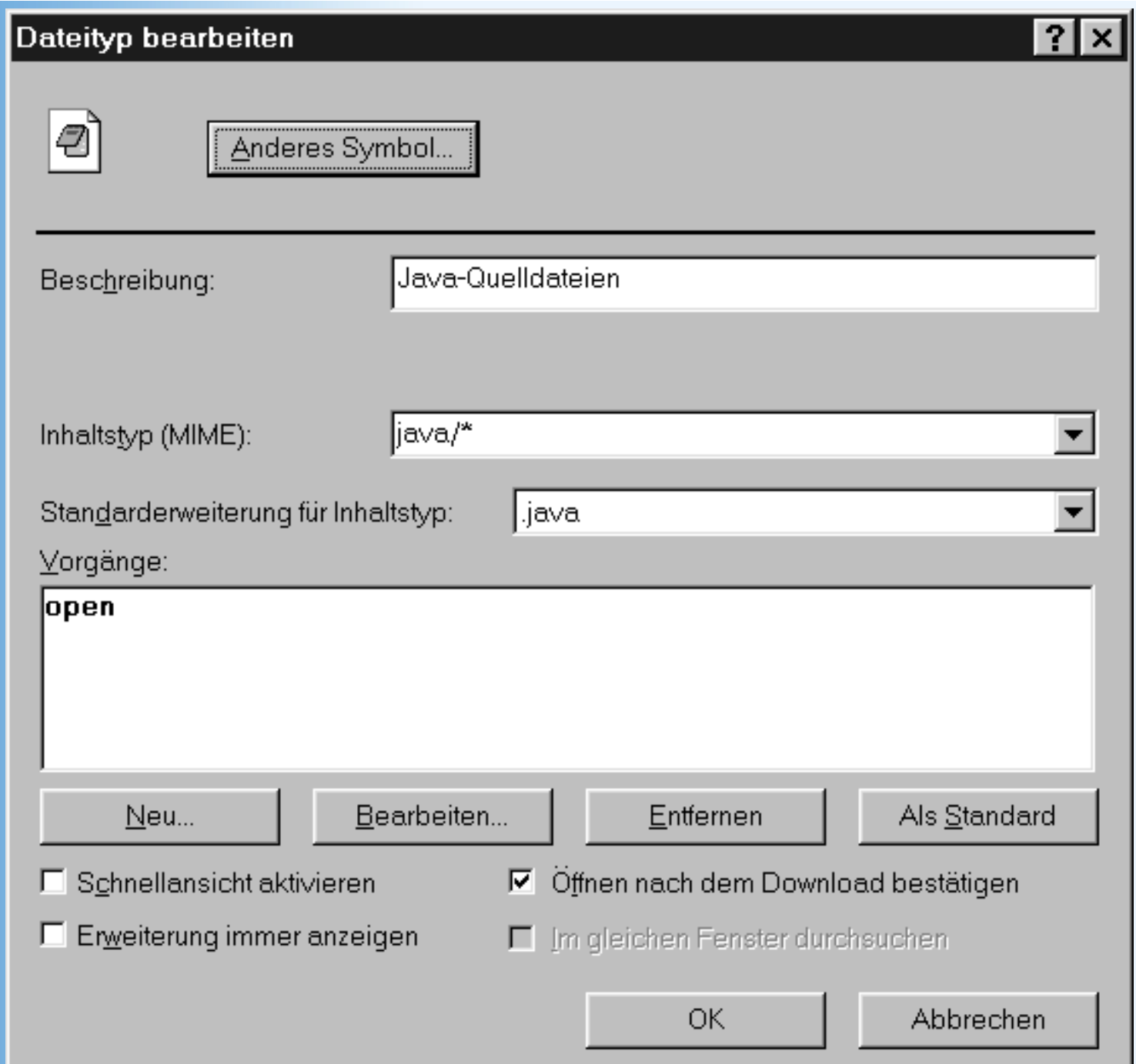


Abbildung E.5:
Das Fenster Dateityp bearbeiten.

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Tag F

Inhalt der CD-ROM zum Buch

Die CD-ROM zu diesem Buch enthält das Java Development Kit für Java 2 von Sun. Außerdem finden Sie hier alle Programmlistings aus dem Buch, einige Shareware- Programme von Drittanbietern, die Internet-Browser MS Internet Explorer und Netscape und einen Schnupperzugang zu T-Online.

Hier der Inhalt der Verzeichnisse im einzelnen:

\JDK

Java 2 SDK von Sun Microsystem für Windows und Solaris. Zur Installation gehen Sie so vor (diesen Schritt können Sie überspringen, wenn Sie bereits eine Java-Entwicklungsumgebung installiert haben, die die neueste Version von Java unterstützt):

1. Die Installationsdatei für das JDK für Windows ist `jdk12-win32.exe`. Zur Installation rufen Sie einfach diese exe-Datei auf. Wir empfehlen, die Dateien in einem Verzeichnis `\jdk` unterzubringen. Wenn Sie über eine ältere Version des JDK verfügen, entfernen Sie diese, bevor Sie die neuere Version installieren.



Wenn das JDK durch ein Setup-Programm installiert wird, bietet das Setup- Programm gewöhnlich ein anderes Standardverzeichnis für jede Version des JDK an, beispielsweise `jdk1.2.3`. Dennoch sollten Sie das Verzeichnis in `jdk` ändern. Wenn Sie allerdings ein Java-Fan sind, der die verschiedenen Versionen des JDK sammelt, übernehmen Sie einfach die vorgegebenen Verzeichnisse.

2. Nehmen Sie das Verzeichnis `jdk\bin` in die PATH-Einstellung auf. Beispielsweise schreiben Sie unter Windows 95 die folgende Zeile an das Ende der Datei `AUTOEXEC.BAT` :

```
SET PATH=C:\jdk\bin;%PATH%<O:P</O:P
```



Wenn Sie das JDK in einem anderen Verzeichnis wie etwa `jdk1.2.3` installiert haben, müssen Sie den Pfad auf `jdk1.2.3\bin` setzen. Die anderen Anweisungen sind sinngemäß zu ändern, ohne daß wir erneut darauf hinweisen.

3. Installieren Sie die Dokumentation (die sich in einem separaten Archiv befindet). Wir empfehlen auch, den Quellcode (der in der Datei `src.jar` im Verzeichnis `jdk` enthalten ist) zu entpacken.



Sun Microsystems gibt häufig aktualisierte Versionen des JDK frei. Auf der Website von Java unter `http://java.sun.com` können Sie nachsehen, ob eine neuere Version des JDK verfügbar ist. In diesem Fall sollten Sie das JDK vom Web herunterladen und anstelle der CD-Version installieren.

\SOURCE

Das Verzeichnis enthält die Beispiele zum Buch in gepackter und ungepackter Form.

\E-Book

Hier finden Sie den kompletten Buchtext in elektornischer Form (HTML-Format). Wenn Sie auf die Datei `Inhalt.htm` klicken, erhalten Sie das Inhaltsverzeichnis des Buches. Die Überschriften des Inhaltsverzeichnisses sind mit den Buchkapiteln verlinkt.

\3RDPARTY

In diesem Verzeichnis finden Sie DemoverSIONen, Share- und Freeware-Programme von Drittanbietern.

\Dienste\CSERVE

Zugangssoftware des Onlinedienstes CompuServe Version 3.02D. Hierin enthalten sind 10 Freistunden zum Kennenlernen dieses Dienstes. Die monatliche Grundgebühr beträgt US\$ 9,95, darin enthalten sind jeden Monat 5 Online-Freistunden. Nach Ablauf dieser 5 Freistunden fallen US\$ 2,95 pro Stunde an, wobei die Abrechnung im Minutentakt erfolgt. Telefongebühren sind darin nicht enthalten.

\Dienste\IE5

Der Internet-Browser Microsoft Internet Explorer 5.0. Mit enthalten ist Outlook Express, Windows Media Player, Netmeeting, Chat 2.5, Wallet und andere Multimedia- Erweiterungen.

\Dienste\Netscape

Der Internet-Browser Netscape Communicator 4.5. Mit enthalten ist das Programm Netscape Composer zum Erstellen von Internet-Seiten.

\Dienste\T-ONLINE

Zugangssoftware des größten europäischen Online-Dienst. Der Grundpreis pro Monat beträgt DM 8,-. Hierin sind 2 Online-Stunden enthalten. Die Verbindungskosten liegen bei 3 Pfennig für das Internet und 3 Pfennig Verbindungsentgeld.

\ESAFE2

eSafe Protect 2.0 Desktop für Windows 95/98 und NT. Die Personal-Desktop-Firewall mit komplettem AntiVirus- und AntiVandal-Schutz!

\KATALOG

Der Markt&Technik-Produktkatalog. Rufen Sie hierfür die Datei `Index.htm` auf.

Hierfür benötigen Sie einen HTML-Browser. Im Verzeichnis `\Dienste\IE5` finden Sie den

Internet Explorer 5.0 von Microsoft.



Bei einem physikalischen Defekt senden Sie bitte die CD mit einer genauen Fehlerbeschreibung an unseren Support. Diese wird Ihnen dann kostenlos ausgetauscht. Vorher sollten Sie die CD aber testen.

Keinen Support können wir Ihnen für Programme von Drittanbietern, sei es Shareware, Freeware oder Demo-Software geben. In diesem Fall wenden Sie sich bitte an den entsprechenden Hersteller des Produktes. Kontaktadressen finden Sie hierbei gewöhnlich in der jeweiligen Readme-Datei.

Shareware-Programme dürfen für eine festgelegte Testzeit kostenlos getestet werden. Nach Ablauf dieser Testzeit müssen Sie diese Programme entweder von der Festplatte löschen oder Sie müssen sich als Benutzer bei dem entsprechenden Hersteller registrieren lassen.

In der Regel ist die Software, die in unseren Büchern besprochen wird, nicht Teil des Lieferumfangs. Bei Fragen dazu wenden Sie sich an den entsprechenden Software-Hersteller.

Wenn Sie mit uns Kontakt aufnehmen, machen Sie bitte folgende Angaben:

- der komplette Buch- oder Softwaretitel
- die ISBN-Nummer
- eine genaue Fehlerbeschreibung
- Ihre Systemkonfiguration
- Ihre Anschrift, Telefonnummer und eventuell Ihre E-Mail-Adresse

Sie erreichen uns unter:

Markt&Technik
Buch- und Software-Verlag GmbH
Technischer Support
Martin-Kollar-Straße 10 - 12
81829 München
Tel.: 089/46003-371
E-Mail: support@pearson.de

Ein Imprint des [Markt+Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0

Stichwortverzeichnis

Symbols

%JAVA_HOME%

- [Die Konfiguration des Java Development Kit](#)

A

Abstract Windowing Toolkit

- [Einfache Benutzeroberflächen für Applets](#)
- [Benutzerschnittstellen entwerfen](#)
- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)
- [Benutzerschnittstellen mit Swing](#)
- [Ereignisbehandlung mit Swing](#)

Abstract Windowing Toolkit (AWT)

- [Bilder, Sound und Animation](#)

abstrakte Methodendefinitionen

- [Pakete, Schnittstellen und mehr](#)

Accessibility-Features

- [Benutzerschnittstellen mit Swing](#)

AccessibleContext-Klasse

- [Benutzerschnittstellen mit Swing](#)

Accessor-Methoden

- [Pakete, Schnittstellen und mehr](#)

action()-Methode

- [Ereignisverarbeitung in Applets](#)
- [Ereignisverarbeitung in Applets](#)
- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)
- [Ereignisbehandlung mit Swing](#)

ActionEvent-Klasse

- [Ereignisbehandlung mit Swing](#)

actionPerformed()-Methode

- [Ereignisbehandlung mit Swing](#)

Activator

- [Eine Handvoll Java](#)

ActiveX

- [Grundlagen der Java-Applets](#)

ActiveX-Steuerelemente

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

ActiveX-Technologie

- [JavaBeans und andere fortgeschrittene Features](#)

add()-Methode

- [Einfache Benutzeroberflächen für Applets](#)
- [Einfache Benutzeroberflächen für Applets](#)
- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)
- [Benutzerschnittstellen mit Swing](#)
- [Benutzerschnittstellen mit Swing](#)

addActionListener()-Methode

- [Ereignisbehandlung mit Swing](#)
- [Ereignisbehandlung mit Swing](#)

addAdjustmentListener()-Methode

- [Ereignisbehandlung mit Swing](#)

addFocusListener()-Methode

- [Ereignisbehandlung mit Swing](#)

addInstance()-Methode

- [Pakete, Schnittstellen und mehr](#)

addItem()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

- [Einfache Benutzeroberflächen für Applets](#)

- [Benutzerschnittstellen mit Swing](#)

addItemListener()-Methode

- [Ereignisbehandlung mit Swing](#)

addKeyListener()-Methode

- [Ereignisbehandlung mit Swing](#)

addline()-Methode

- [Ereignisverarbeitung in Applets](#)

addMotionListener()-Methode

- [Ereignisbehandlung mit Swing](#)

addMouseListener()-Methode

- [Ereignisbehandlung mit Swing](#)

addspot()-Methode

- [Ereignisverarbeitung in Applets](#)

addWindowListener()-Methode

- [Ereignisbehandlung mit Swing](#)

AdjustmentEvent-Klasse

- [Ereignisbehandlung mit Swing](#)

Adjustment-Klasse

- [Ereignisbehandlung mit Swing](#)

AdjustmentListener-Schnittstelle

- [Ereignisbehandlung mit Swing](#)

adjustmentValueChanged()-Methode

- [Ereignisbehandlung mit Swing](#)

Aktionsergebnisse

- [Ereignisverarbeitung in Applets](#)

Behandlung



[Ereignisverarbeitung in Applets](#)

Aktualisieren, Panels

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Animation

Flimmern reduzieren



[Bilder, Sound und Animation](#)

Grundlagen



[Bilder, Sound und Animation](#)

mit Bildern



[Bilder, Sound und Animation](#)

Anweisung



[Das Java-ABC](#)

Blockanweisung



[Arrays, Bedingungen und Schleifen](#)

break



[Arrays, Bedingungen und Schleifen](#)



[Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

catch



[Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Continue



[Arrays, Bedingungen und Schleifen](#)

extends



[Objektorientierte Programmierung - ein erster Eindruck](#)

finally



[Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

if...else



[Arrays, Bedingungen und Schleifen](#)

implements



[Bilder, Sound und Animation](#)



[Pakete, Schnittstellen und mehr](#)

import

■ [Objektorientierte Programmierung - ein erster Eindruck](#)



[Pakete, Schnittstellen und mehr](#)

new

■ [Objektorientierte Programmierung - ein erster Eindruck](#)

package



[Pakete, Schnittstellen und mehr](#)

switch



[Arrays, Bedingungen und Schleifen](#)

throws

■ [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

try

■ [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Anwendungen

Client



[Kommunikation über das Internet](#)

Entwerfen



[Kommunikation über das Internet](#)

RMI erstellen

■ [JavaBeans und andere fortgeschrittene Features](#)

■ [JavaBeans und andere fortgeschrittene Features](#)

Trivia



[Kommunikation über das Internet](#)



[Kommunikation über das Internet](#)

Anzeigen

Applet-Informationen

■ [JavaBeans und andere fortgeschrittene Features](#)

Fenster

■ [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

APIs, JavaBeans

- [JavaBeans und andere fortgeschrittene Features](#)

Applet

-

[Eine Handvoll Java](#)

auf Web-Seite einfügen

-

[Grundlagen der Java-Applets](#)

Datenstrom öffnen

-

[Java und Streams](#)

-

[Java und Streams](#)

digitale Signaturen

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

erstellen

-

[Grundlagen der Java-Applets](#)

GetRaven

-

[Kommunikation über das Internet](#)

-

[Kommunikation über das Internet](#)

im Web anbieten

-

[Grundlagen der Java-Applets](#)

Informationen anzeigen

- [JavaBeans und andere fortgeschrittene Features](#)

initialisieren

-

[Grundlagen der Java-Applets](#)

Inter-Applet-Kommunikation

- [JavaBeans und andere fortgeschrittene Features](#)

- [JavaBeans und andere fortgeschrittene Features](#)

mehrere Web-Seiten

- [JavaBeans und andere fortgeschrittene Features](#)

Parameter

-

[Grundlagen der Java-Applets](#)

prüfen

- [Grundlagen der Java-Applets](#)

Sicherheitseinschränkungen

- [Grundlagen der Java-Applets](#)

starten

- [Grundlagen der Java-Applets](#)
- [Bilder, Sound und Animation](#)

stoppen

- [Grundlagen der Java-Applets](#)
- [Bilder, Sound und Animation](#)

Verknüpfungen erstellen

- [JavaBeans und andere fortgeschrittene Features](#)

zeichnen

- [Grundlagen der Java-Applets](#)

zerstören

- [Grundlagen der Java-Applets](#)

AppletContext-Klasse

- [JavaBeans und andere fortgeschrittene Features](#)

Applet-Klasse

- [Benutzerschnittstellen entwerfen](#)

APPLET-Tag

- [Grundlagen der Java-Applets](#)

Appletviewer

- [Grundlagen der Java-Applets](#)
- [Grundlagen der Java-Applets](#)

Application Builder Support

- [JavaBeans und andere fortgeschrittene Features](#)

Applikation

- [Eine Handvoll Java](#)
- [Java: Eine Klassensprache](#)

Archive in Java

- [Grundlagen der Java-Applets](#)

Archiv-Formate

- [Grundlagen der Java-Applets](#)
- Cabinet
 - [Grundlagen der Java-Applets](#)
- Zip
 - [Grundlagen der Java-Applets](#)

Argumente verarbeiten

- [Java: Eine Klassensprache](#)

Array

- [Arrays, Bedingungen und Schleifen](#)
- Deklarieren
 - [Arrays, Bedingungen und Schleifen](#)
- Erstellen
 - [Arrays, Bedingungen und Schleifen](#)
- Klasse
 - [Arbeiten mit Objekten](#)
- Multidimensional
 - [Arrays, Bedingungen und Schleifen](#)

Asterisk (*)

- [Pakete, Schnittstellen und mehr](#)

Attribute

- [Objektorientierte Programmierung - ein erster Eindruck](#)

AU-Dateien



[Bilder, Sound und Animation](#)

Audio-Formate

AIFF



[Bilder, Sound und Animation](#)

AU



[Bilder, Sound und Animation](#)

WAF



[Bilder, Sound und Animation](#)

Ausdruck



[Das Java-ABC](#)



[Das Java-ABC](#)

Ausnahmen

Arten



[Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Auswerfen



[Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Designaspekte



[Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Einschränkungen



[Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

EOFException



[Java und Streams](#)

Handler



[Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

IOException



[Java und Streams](#)

throws



[Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

UTFDataFormatException



[Java und Streams](#)

Auswahlereignisse

- [Ereignisverarbeitung in Applets](#)

Authenicode

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

autoexec.bat

- [Die Konfiguration des Java Development Kit](#)

available()-Methode

-

[Java und Streams](#)

AWT

- [Einfache Benutzeroberflächen für Applets](#)

Ausschneiden/Kopieren/Einfügen-Unterstützung

- [JavaBeans und andere fortgeschrittene Features](#)

übersicht

- [Einfache Benutzeroberflächen für Applets](#)

Komponenten

- [Einfache Benutzeroberflächen für Applets](#)

B

BaseFrame1-Klasse

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

BaseFrame2-Klasse

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Bedingung

- [Arrays, Bedingungen und Schleifen](#)

Bedingungsoperator, Definition

- [Arrays, Bedingungen und Schleifen](#)

Befehlszeilenargumente

- [Java: Eine Klassensprache](#)

Benutzerkoordinatensystem

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Bgen

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Bier

- [Objektorientierte Programmierung - ein erster Eindruck](#)

Bill Joy

- [Eine Handvoll Java](#)

BinaryTree-Klasse

- [Pakete, Schnittstellen und mehr](#)

Bjarne Stroustrup

- [Eine Handvoll Java](#)

Blockanweisung

- [Arrays, Bedingungen und Schleifen](#)

boolesche Werte

- [Objektorientierte Programmierung - ein erster Eindruck](#)

BorderLayout-Klasse

- [Benutzerschnittstellen entwerfen](#)
- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Borland JBuilder

- [Eine Handvoll Java](#)

BritishCars-Klasse

- [Pakete, Schnittstellen und mehr](#)

BritishMotorcycle-Klasse

- [Pakete, Schnittstellen und mehr](#)

BritishThing-Klasse

- [Pakete, Schnittstellen und mehr](#)

Browser

Statuszeile, Meldungen anzeigen

- [JavaBeans und andere fortgeschrittene Features](#)

Web-Verbindungen erstellen

- [Kommunikation über das Internet](#)
- [Kommunikation über das Internet](#)

BSD

- [Kommunikation über das Internet](#)

BufferedInputStream-Klasse

- [Java und Streams](#)

BufferedOutputStream-Klasse

- [Java und Streams](#)

BufferedReader-Klasse

- [Java und Streams](#)

BufferedWriter-Klasse

- [Java und Streams](#)

buildConstraints()-Methode

- [Benutzerschnittstellen entwerfen](#)

ButtonGroup-Klasse

- [Benutzerschnittstellen mit Swing](#)

Button-Klasse

- [Einfache Benutzeroberflächen für Applets](#)
- [Benutzerschnittstellen mit Swing](#)

ButtonLink-Klasse

- [JavaBeans und andere fortgeschrittene Features](#)

Byteanzahl ausgeben

- [Java und Streams](#)

Byte-Array erstellen

-

[Java und Streams](#)

ByteArrayInputStream-Klasse

-

[Java und Streams](#)

ByteArrayOutputStream-Klasse

-

[Java und Streams](#)

Bytecode

-

[Eine Handvoll Java](#)

Bytecode-Interpreter

-

[Eine Handvoll Java](#)

C

C

-

[Eine Handvoll Java](#)

C++

-

[Eine Handvoll Java](#)

CAB-Datei

-

[Grundlagen der Java-Applets](#)

Canvas-Klasse

- [Einfache Benutzeroberflächen für Applets](#)

CardLayout-Klasse

-

[Benutzerschnittstellen entwerfen](#)

Casting

-

[Arbeiten mit Objekten](#)

Objekte



[Arbeiten mit Objekten](#)

Primitivtypen



[Arbeiten mit Objekten](#)

certificate

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

certificate authority

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

changeColor()-Methode

- [Ereignisverarbeitung in Applets](#)

CharArrayReader-Klasse

- [Java und Streams](#)

CharArrayWriter-Klasse

- [Java und Streams](#)

CheckboxGroup-Klasse

- [Einfache Benutzeroberflächen für Applets](#)
- [Benutzerschnittstellen mit Swing](#)

Checkbox-Klasse

- [Einfache Benutzeroberflächen für Applets](#)

CheckBoxMenuItem-Klasse

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Choice-Klasse

- [Einfache Benutzeroberflächen für Applets](#)
- [Einfache Benutzeroberflächen für Applets](#)
- [Benutzerschnittstellen mit Swing](#)

Claris Homepage

- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Grundlagen der Java-Applets](#)

classes.zip

- [Die Konfiguration des Java Development Kit](#)

Class-not-found-Fehler

- [Die Konfiguration des Java Development Kit](#)
- [Die Konfiguration des Java Development Kit](#)

CLASSPATH

-

[Java: Eine Klassensprache](#)

CLASSPATH-Einstellung

- [Die Konfiguration des Java Development Kit](#)

JDK-Ordner nicht enthalten

- [Die Konfiguration des Java Development Kit](#)

ndern

- [Die Konfiguration des Java Development Kit](#)

nicht vorhanden

- [Die Konfiguration des Java Development Kit](#)

prüfen

- [Die Konfiguration des Java Development Kit](#)

clickCount-Variable

- [Ereignisverarbeitung in Applets](#)

Clint Eastwood

- [Eine Handvoll Java](#)

Clipboard-Klasse

- [JavaBeans und andere fortgeschrittene Features](#)

close()-Methode

- [Java und Streams](#)
- [Java und Streams](#)
- [Java und Streams](#)

Color Space

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

ColorControls-Klasse

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)
- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Color-Klasse

- [Ereignisverarbeitung in Applets](#)
- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

ColorTest-Klasse

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Command

- [Ereignisverarbeitung in Applets](#)

Common Object Request Broker Architecture

- [JavaBeans und andere fortgeschrittene Features](#)

Component-Klasse

- [Ereignisverarbeitung in Applets](#)
- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Constructor-Klasse

- [Arbeiten mit Objekten](#)

content pane

- [Benutzerschnittstellen mit Swing](#)

controlDown()-Methode

- [Ereignisverarbeitung in Applets](#)

CORBA

- [JavaBeans und andere fortgeschrittene Features](#)

CountInstances-Klasse

- [Pakete, Schnittstellen und mehr](#)

countItems()-Methode

- [Einfache Benutzeroberflächen für Applets](#)
- [Einfache Benutzeroberflächen für Applets](#)

cut-and-paste-Funktionalitt

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

D

Daniel Griscom

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

DataFlavor-Klasse

- [JavaBeans und andere fortgeschrittene Features](#)

DataInputStream-Klasse

- [Java und Streams](#)



[Java und Streams](#)

DataOutputStream-Klasse



[Java und Streams](#)

Dateierweiterung



[Texteditoren und das JDK](#)

Dateitypen

bestehende Verknüpfung löschen



[Texteditoren und das JDK](#)

neuen Typ registrieren



[Texteditoren und das JDK](#)

registrieren



[Texteditoren und das JDK](#)

Symbol zuweisen



[Texteditoren und das JDK](#)

Datenbank, SQL

- [JavaBeans und andere fortgeschrittene Features](#)

Datenbertragung

MIME-Typen



[JavaBeans und andere fortgeschrittene Features](#)

Objekte erstellen



[JavaBeans und andere fortgeschrittene Features](#)



[JavaBeans und andere fortgeschrittene Features](#)

Operationen der Zwischenablage



[JavaBeans und andere fortgeschrittene Features](#)



[JavaBeans und andere fortgeschrittene Features](#)

Spezifikation, Web-Standort



[JavaBeans und andere fortgeschrittene Features](#)

Unterstützung



[JavaBeans und andere fortgeschrittene Features](#)

Datenströme

Anfangskapazitt



[Java und Streams](#)

Ausgabe



[Java und Streams](#)

Ausgabe verschachteln



[Java und Streams](#)

BufferedInputStream



[Java und Streams](#)

BufferedOutputStream



[Java und Streams](#)

BufferedReader



[Java und Streams](#)

BufferedWriter



[Java und Streams](#)

ByteArrayOutputStream



[Java und Streams](#)

CharArrayWriter



[Java und Streams](#)

Dateiverarbeitung



[Java und Streams](#)

Definition



[Java und Streams](#)

Eingaben



[Java und Streams](#)

EOFException



[Java und Streams](#)

erneut lesen



[Java und Streams](#)

FileDescriptor



[Java und Streams](#)



[Java und Streams](#)

FileOutputStream



[Java und Streams](#)

FileWriter



[Java und Streams](#)

FilterOutputStream



[Java und Streams](#)

FilterWriter



[Java und Streams](#)

Garbage-Collector



[Java und Streams](#)

getFD()



[Java und Streams](#)

Kompilierzeit



[Java und Streams](#)

Laufzeit



[Java und Streams](#)

ObjectInput



[Java und Streams](#)

ObjectOutputStream



[Java und Streams](#)



[Java und Streams](#)

PrintStream



[Java und Streams](#)

Reader



[Java und Streams](#)

schließen



[Java und Streams](#)

SequenceInputStream



[Java und Streams](#)

StringBufferInputStream



[Java und Streams](#)

StringReader



[Java und Streams](#)

Verketteten



[Java und Streams](#)

Writer



[Java und Streams](#)



[Java und Streams](#)

zurücksetzen



[Java und Streams](#)

Datentypen



[Das Java-ABC](#)

DCOM

- [JavaBeans und andere fortgeschrittene Features](#)

Delphi



[Eine Handvoll Java](#)

destroy()-Methode

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Dezimal-System



[Das Java-ABC](#)

Dialogfelder

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)
- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Erstellen

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)
- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Modal

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Schaltflächen erstellen

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Dialog-Klasse

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Digitale Signaturen

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

disable()-Methode

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Distributed Component Object Model

- [JavaBeans und andere fortgeschrittene Features](#)

Dither-Muster

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Domino-Applet

- [Eine Handvoll Java](#)

Doppelklicks mit der Maus

- [Ereignisverarbeitung in Applets](#)

Doppelte Pufferung

- [Bilder, Sound und Animation](#)

do-Schleife

- [Arrays, Bedingungen und Schleifen](#)

Drag&Drop

- [Eine Handvoll Java](#)

Dubbed Metal

- [Benutzerschnittstellen mit Swing](#)

Duke

- [Kommunikation über das Internet](#)

E

Ebenen, RMI

- [JavaBeans und andere fortgeschrittene Features](#)

Eckeinsteze (Insets)



- [Benutzerschnittstellen entwerfen](#)

Eigenschaften, Hinzufügen



- [Mehr über Methoden](#)

Einfachvererbung



- [Objektorientierte Programmierung - ein erster Eindruck](#)

Eingabestrom berspringen



- [Java und Streams](#)

electricalSystemFailure()-Methode



- [Pakete, Schnittstellen und mehr](#)

elementAt()-Methode



- [Java und Streams](#)

Elementfunktion



- [Objektorientierte Programmierung - ein erster Eindruck](#)

Ellipsen



- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

enable()-Methode



- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Entwerfen, Anwendungen



- [Kommunikation über das Internet](#)

Ereignis

- Behandlung



- [Ereignisverarbeitung in Applets](#)

- Doppelklicks



- [Ereignisverarbeitung in Applets](#)

Ergnzungstasten



- [Ereignisverarbeitung in Applets](#)

Eric Carroll



- [Eine Handvoll Java](#)

Error-Klasse

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Escapecodes

-

[Das Java-ABC](#)

Event Listener

- [Ereignisbehandlung mit Swing](#)
- [Ereignisbehandlung mit Swing](#)

ActionListener

- [Ereignisbehandlung mit Swing](#)

AdjustmentListener

- [Ereignisbehandlung mit Swing](#)

FocusListener

- [Ereignisbehandlung mit Swing](#)

ItemListener

- [Ereignisbehandlung mit Swing](#)

KeyListener

- [Ereignisbehandlung mit Swing](#)

MouseListener

- [Ereignisbehandlung mit Swing](#)

MouseMotionListener

- [Ereignisbehandlung mit Swing](#)

WindowListener

- [Ereignisbehandlung mit Swing](#)

Event-Klasse

- [Ereignisverarbeitung in Applets](#)
- [Ereignisverarbeitung in Applets](#)

- [Ereignisverarbeitung in Applets](#)
 - Standardtasten
 - [Ereignisverarbeitung in Applets](#)

EventObject-Klasse

- [Ereignisbehandlung mit Swing](#)

Exception-Klasse

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Expression

- [Das Java-ABC](#)

F

Farbbeschreibungssystem

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Farbe

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Farbraum

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Farbtheorie

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Fenster

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)
 - Erstellen Anzeigen
 - [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Fensterereignisse

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)
- [Ereignisbehandlung mit Swing](#)

ffentlicher Schlssel

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Field-Klasse

- [Arbeiten mit Objekten](#)

FileDialog-Klasse

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

FileInputStream-Klasse

○

[Java und Streams](#)

FileOutputStream-Klasse

○

[Java und Streams](#)

FileReader-Klasse

○

[Java und Streams](#)

○

[Java und Streams](#)

FileWriter-Klasse

○

[Java und Streams](#)

○

[Java und Streams](#)

FilterInputStream-Klasse

○

[Java und Streams](#)

FilterOutputStream-Klasse

○

[Java und Streams](#)

FilterReader-Klasse

○

[Java und Streams](#)

FilterWriter-Klasse

○

[Java und Streams](#)

finalize()-Methode

○

[Java und Streams](#)

Fishhead Software

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Flavor

- [JavaBeans und andere fortgeschrittene Features](#)

Flmmuster

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

FlowLayout-Klasse

- [Benutzerschnittstellen entwerfen](#)
- [Benutzerschnittstellen entwerfen](#)

flush()-Methode

- [Java und Streams](#)
- [Java und Streams](#)
- [Java und Streams](#)

FocusEvent-Klasse

- [Ereignisbehandlung mit Swing](#)

focusGained()-Methode

- [Ereignisbehandlung mit Swing](#)

FocusListener-Schnittstelle

- [Ereignisbehandlung mit Swing](#)

focusLost()-Methode

- [Ereignisbehandlung mit Swing](#)

Fokusereignisse

- [Ereignisverarbeitung in Applets](#)
Behandlung
 - [Ereignisverarbeitung in Applets](#)

Formate, Datenbertragungsobjekte

- [JavaBeans und andere fortgeschrittene Features](#)

for-Schleife

- [Arrays, Bedingungen und Schleifen](#)

Frame-Klasse

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Frames

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Erstellen

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Fruit-Klasse

- [Pakete, Schnittstellen und mehr](#)

Fruitlike

- [Pakete, Schnittstellen und mehr](#)

G

Gamelan

- [Eine Handvoll Java](#)

Garbage Collector

- [Arbeiten mit Objekten](#)
- [Mehr über Methoden](#)
- [Grundlagen der Java-Applets](#)

finalize()

■

[Java und Streams](#)

George Boole

- [Objektorientierte Programmierung - ein erster Eindruck](#)

Gertekoordinatensystem

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

getAccessibleContext()-Methode

- [Benutzerschnittstellen mit Swing](#)

getActionCommand()-Methode



[Ereignisbehandlung mit Swing](#)

getAdjustmentType()-Methode



[Ereignisbehandlung mit Swing](#)

getAppletContext()-Methode



[Kommunikation über das Internet](#)



[JavaBeans und andere fortgeschrittene Features](#)



[JavaBeans und andere fortgeschrittene Features](#)

getApplets()-Methode



[JavaBeans und andere fortgeschrittene Features](#)

getClickCount()-Methode



[Ereignisbehandlung mit Swing](#)

getContents()-Methode



[JavaBeans und andere fortgeschrittene Features](#)

getCrossPlatformLookAndFeelClassName()-Methode



[Benutzerschnittstellen mit Swing](#)

getCurrent()-Methode



[Einfache Benutzeroberflächen für Applets](#)

getFD()-Methode



[Java und Streams](#)



[Java und Streams](#)

getHSBColor()-Methode



[Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

getInputStream()-Methode



[Kommunikation über das Internet](#)

getInsets()-Methode



[Benutzerschnittstellen entwerfen](#)



[Fortgeschrittene Benutzeroberflächen mit dem AWT](#)



[Fortgeschrittene Benutzeroberflächen mit dem AWT](#)



[Benutzerschnittstellen mit Swing](#)

getItem()-Methode

- [Einfache Benutzeroberflächen für Applets](#)
- [Einfache Benutzeroberflächen für Applets](#)
-

[Ereignisbehandlung mit Swing](#)

getItemCount()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

getKeyChar()-Klasse

-
- [Ereignisbehandlung mit Swing](#)

getKeyChar()-Methode

-
- [Ereignisbehandlung mit Swing](#)

getLabel()-Methode

- [Einfache Benutzeroberflächen für Applets](#)
-
- [Ereignisverarbeitung in Applets](#)

getNumInstances()-Methode

-
- [Pakete, Schnittstellen und mehr](#)

getParameter()-Methode

-
- [Grundlagen der Java-Applets](#)

getParent()-Methode

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

getPoint()-Methode

-
- [Ereignisbehandlung mit Swing](#)

GetRaven-Applet

-
- [Kommunikation über das Internet](#)
-
- [Kommunikation über das Internet](#)

GetRaven-Klasse (Listing)

-
- [Kommunikation über das Internet](#)

getSelectedIndex()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

- [Einfache Benutzeroberflächen für Applets](#)

getSelectedIndexes()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

getSelectedItem()-Methode

- [Einfache Benutzeroberflächen für Applets](#)
- [Einfache Benutzeroberflächen für Applets](#)

getSelectedItems()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

getSource()-Methode

- [Ereignisbehandlung mit Swing](#)

getState()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

getStateChange()-Methode

- [Ereignisbehandlung mit Swing](#)

getSystemClipboard()-Methode

- [JavaBeans und andere fortgeschrittene Features](#)

getSystemLookAndFeelClassName()-Methode

- [Benutzerschnittstellen mit Swing](#)

getText()-Methode

- [Einfache Benutzeroberflächen für Applets](#)
- [Einfache Benutzeroberflächen für Applets](#)
- [Einfache Benutzeroberflächen für Applets](#)

getToolkit()-Methode

- [JavaBeans und andere fortgeschrittene Features](#)

getTransferData()-Methode

- [JavaBeans und andere fortgeschrittene Features](#)
- [JavaBeans und andere fortgeschrittene Features](#)

getTransferDataFlavors()-Methode

- [JavaBeans und andere fortgeschrittene Features](#)
- [JavaBeans und andere fortgeschrittene Features](#)

getValue()-Methode

- [Einfache Benutzeroberflächen für Applets](#)
- [Ereignisbehandlung mit Swing](#)

getY()-Methode



[Ereignisbehandlung mit Swing](#)

Globale Variablen



[Das Java-ABC](#)

Gltigkeitsbereich



[Java: Eine Klassesprache](#)

gotFocus()-Methode



[Ereignisverarbeitung in Applets](#)

Graphics-Klasse



[Ereignisverarbeitung in Applets](#)

Green-Projekt



[Eine Handvoll Java](#)

GridBagConstraints-Klasse



[Benutzerschnittstellen entwerfen](#)



[Benutzerschnittstellen entwerfen](#)

GridLayout-Klasse



[Benutzerschnittstellen entwerfen](#)

GridLayout-Klasse



[Benutzerschnittstellen entwerfen](#)

H

handleChoice()-Methode



[Ereignisverarbeitung in Applets](#)

handleEvent()-Methode



[Ereignisverarbeitung in Applets](#)



[Ereignisverarbeitung in Applets](#)



[Ereignisverarbeitung in Applets](#)

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)
- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

handleText()-Methode

- [Ereignisverarbeitung in Applets](#)

Hauptsystemthread

- [Bilder, Sound und Animation](#)

Hexadezimal-System

- [Das Java-ABC](#)

hide()-Methode

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Hierarchieform bei Paketen

- [Pakete, Schnittstellen und mehr](#)

Hilfsklassen

- [Java: Eine Klassensprache](#)

Hintergrundfarbe eines Applet

- [Bilder, Sound und Animation](#)

Hilfklassen

- [Arbeiten mit Objekten](#)

hostile applets

- [Grundlagen der Java-Applets](#)

HotJava

- [Eine Handvoll Java](#)
- [Java: Eine Klassensprache](#)
- [Grundlagen der Java-Applets](#)

I

Icon

-

[Benutzerschnittstellen mit Swing](#)

Icon-Klasse

- [Benutzerschnittstellen mit Swing](#)

Identifikation von Klassen

- [Pakete, Schnittstellen und mehr](#)

id-Instanzvariable

- [Ereignisverarbeitung in Applets](#)

Image-Klasse

- [Benutzerschnittstellen mit Swing](#)

implements-Anweisung

- [Pakete, Schnittstellen und mehr](#)

import-Anweisung

- [Pakete, Schnittstellen und mehr](#)

init()-Methode

- [Grundlagen der Java-Applets](#)

initQnA()-Methode

- [Kommunikation über das Internet](#)

InputStream-Klasse

- [Java und Streams](#)

insert()-Methode

- [Einfache Benutzeroberflächen für Applets](#)
- [Pakete, Schnittstellen und mehr](#)

insertObject()-Methode

- [Pakete, Schnittstellen und mehr](#)

insertText()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

Insets (Eckeinste)

-

[Benutzerschnittstellen entwerfen](#)

insets()-Methode

- [Benutzerschnittstellen entwerfen](#)

Insets-Klasse

- [Benutzerschnittstellen entwerfen](#)

Instanz

- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Objektorientierte Programmierung - ein erster Eindruck](#)

Instanzmethode

- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Java: Eine Klassensprache](#)

Instanzvariable

- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Objektorientierte Programmierung - ein erster Eindruck](#)
-

[Das Java-ABC](#)

- [Arbeiten mit Objekten](#)
- [Grundlagen der Java-Applets](#)

Definieren

- [Java: Eine Klassensprache](#)

Zugriffskontrolle

- [Pakete, Schnittstellen und mehr](#)

Inter-Applet-Kommunikation

- [JavaBeans und andere fortgeschrittene Features](#)

Interface

- [Objektorientierte Programmierung - ein erster Eindruck](#)

Interne Klassen

- [Pakete, Schnittstellen und mehr](#)

Internet

Plattformabhngigkeit

- [JavaBeans und andere fortgeschrittene Features](#)

Verbindungen erstellen

- [Kommunikation über das Internet](#)
- [Kommunikation über das Internet](#)

Introspection, siehe Reflection

- [Arbeiten mit Objekten](#)

Introspektion

- [JavaBeans und andere fortgeschrittene Features](#)

isDataFlavorSupported()-Methode

- [JavaBeans und andere fortgeschrittene Features](#)

isEditable()-Methode

- [Einfache Benutzeroberflächen für Applets](#)
- [Einfache Benutzeroberflächen für Applets](#)

Item-Ereignisse

- [Ereignisbehandlung mit Swing](#)

ItemEvent-Klasse

- [Ereignisbehandlung mit Swing](#)

ItemListener-Schnittstelle

- [Ereignisbehandlung mit Swing](#)

itemStateChanged()-Methode

- [Ereignisbehandlung mit Swing](#)

J

Jabberwock

- [Objektorientierte Programmierung - ein erster Eindruck](#)

James Gosling

- [Eine Handvoll Java](#)

jar

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

JAR-Datei

-

[Grundlagen der Java-Applets](#)

jarsigner

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Java

Anwendungen



[Java: Eine Klassensprache](#)

API-Beschreibung



[Java und Streams](#)

Applets



[Grundlagen der Java-Applets](#)

Applikationen



[Java: Eine Klassensprache](#)

Archive



[Grundlagen der Java-Applets](#)

Datenströme



[Java und Streams](#)

Homepage



[Eine Handvoll Java](#)

im Vergleich zu JavaBeans



[JavaBeans und andere fortgeschrittene Features](#)

Koordinatensystem



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

Pointer/Zeiger



[Arbeiten mit Objekten](#)

Referenzen



[Arbeiten mit Objekten](#)

Schutzebenen



[Pakete, Schnittstellen und mehr](#)

Speicherverwaltung



- [Arbeiten mit Objekten](#)

Versionen



- [Eine Handvoll Java](#)

Java Developer's Kit



- [Eine Handvoll Java](#)

Java Development Kit



- [Die Konfiguration des Java Development Kit](#)



- [Texteditoren und das JDK](#)

Konfiguration unter Unix



- [Die Konfiguration des Java Development Kit](#)

Konfiguration unter Windows



- [Die Konfiguration des Java Development Kit](#)

Java Foundation Classes



- [Benutzerschnittstellen mit Swing](#)

Java Laufzeitumgebung



- [Eine Handvoll Java](#)

Java Runtime Environment



- [Eine Handvoll Java](#)

Java Virtual Machine



- [Eine Handvoll Java](#)

java.awt.image.ImageConsumer



- [Pakete, Schnittstellen und mehr](#)

java.awt.imageProducer



- [Pakete, Schnittstellen und mehr](#)

java.lang.Runnable



- [Pakete, Schnittstellen und mehr](#)

java.util Enumeration



[Pakete, Schnittstellen und mehr](#)

java.util.Observable

- [Pakete, Schnittstellen und mehr](#)

Java2D

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Java2D, Darstellungsattribute festlegen

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

JavaBeans

- [JavaBeans und andere fortgeschrittene Features](#)
- [JavaBeans und andere fortgeschrittene Features](#)
- [JavaBeans und andere fortgeschrittene Features](#)

API

- [JavaBeans und andere fortgeschrittene Features](#)
- [JavaBeans und andere fortgeschrittene Features](#)

Designziele

- [JavaBeans und andere fortgeschrittene Features](#)
- [JavaBeans und andere fortgeschrittene Features](#)
- [JavaBeans und andere fortgeschrittene Features](#)

im Vergleich zu Java

- [JavaBeans und andere fortgeschrittene Features](#)

Java-Klassenbibliothek

- [Pakete, Schnittstellen und mehr](#)
-

[Java und Streams](#)

JavaScript

- [Eine Handvoll Java](#)
- [Eine Handvoll Java](#)

JavaSoft

- [Eine Handvoll Java](#)

Web-Seite

- [JavaBeans und andere fortgeschrittene Features](#)

JButton-Klasse

-

[Benutzerschnittstellen mit Swing](#)



[Benutzerschnittstellen mit Swing](#)

JCheckBox-Klasse



[Benutzerschnittstellen mit Swing](#)

JComboBox-Klasse



[Benutzerschnittstellen mit Swing](#)

JComponent-Klasse



[Benutzerschnittstellen mit Swing](#)

JDBC



[JavaBeans und andere fortgeschrittene Features](#)



[JavaBeans und andere fortgeschrittene Features](#)

Web-Standorte

- [JavaBeans und andere fortgeschrittene Features](#)

JDK



[Eine Handvoll Java](#)



[Grundlagen der Java-Applets](#)

Installation unter Solaris



[Eine Handvoll Java](#)

Installation unter Windows



[Eine Handvoll Java](#)

Serialisation



[Java und Streams](#)

Jeffrey Zeldman



[Benutzerschnittstellen mit Swing](#)

JFrame-Klasse



[Benutzerschnittstellen mit Swing](#)



[Ereignisbehandlung mit Swing](#)

JLabel-Klasse

- [Benutzerschnittstellen mit Swing](#)

Jonathan Payne

- [Eine Handvoll Java](#)

JOptionPane-Klasse

- [Benutzerschnittstellen mit Swing](#)
- [Benutzerschnittstellen mit Swing](#)

JPanel-Klasse

- [Benutzerschnittstellen mit Swing](#)

JPasswordField-Klasse

- [Benutzerschnittstellen mit Swing](#)

JRadioButton-Klasse

- [Benutzerschnittstellen mit Swing](#)

JScrollBar-Klasse

- [Benutzerschnittstellen mit Swing](#)

JTextArea-Klasse

- [Benutzerschnittstellen mit Swing](#)

JTextField-Klasse

- [Benutzerschnittstellen mit Swing](#)

Just-In-Time-Compiler

- [Eine Handvoll Java](#)

JVM

- [Eine Handvoll Java](#)

JWindow-Klasse

- [Ereignisbehandlung mit Swing](#)

keyDown()-Methode

- [Ereignisverarbeitung in Applets](#)
- [Ereignisbehandlung mit Swing](#)

KeyEvent-Klasse

- [Ereignisbehandlung mit Swing](#)

KeyListener-Schnittstelle

- [Ereignisbehandlung mit Swing](#)

keyPressed()-Methode

- [Ereignisbehandlung mit Swing](#)

keyReleased()-Methode

- [Ereignisbehandlung mit Swing](#)

keytool

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

keyTyped()-Methode

- [Ereignisbehandlung mit Swing](#)

keyUp()-Methode

- [Ereignisverarbeitung in Applets](#)

Klasse

- [Objektorientierte Programmierung - ein erster Eindruck](#)

abstract



[Pakete, Schnittstellen und mehr](#)

AccessibleContext



[Benutzerschnittstellen mit Swing](#)

ActionEvent



[Ereignisbehandlung mit Swing](#)

Adjustment



[Ereignisbehandlung mit Swing](#)

AdjustmentEvent



[Ereignisbehandlung mit Swing](#)

Applet



[Grundlagen der Java-Applets](#)



[Benutzerschnittstellen entwerfen](#)

AppletContext



[JavaBeans und andere fortgeschrittene Features](#)

Arc2D.Float



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

Array



[Arbeiten mit Objekten](#)

Attribute



[Objektorientierte Programmierung - ein erster Eindruck](#)

AudioClip



[Bilder, Sound und Animation](#)

BaseFrame1



[Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

BaseFrame2



[Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

BasicStroke



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

Bestandteile



[Objektorientierte Programmierung - ein erster Eindruck](#)

BinaryTree



[Pakete, Schnittstellen und mehr](#)

BorderLayout



[Benutzerschnittstellen entwerfen](#)



[Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

BritishCars



[Pakete, Schnittstellen und mehr](#)

BritishMotorcycle



[Pakete, Schnittstellen und mehr](#)

BritishThing



[Pakete, Schnittstellen und mehr](#)

BufferedInputStream



[Java und Streams](#)

BufferedOutputStream



[Java und Streams](#)

BufferedWriter



[Java und Streams](#)

Button



[Einfache Benutzeroberflächen für Applets](#)



[Benutzerschnittstellen mit Swing](#)

ButtonGroup



[Benutzerschnittstellen mit Swing](#)

ButtonLink



[JavaBeans und andere fortgeschrittene Features](#)

ByteArrayInputStream



[Java und Streams](#)

ByteArrayOutputStream



[Java und Streams](#)

Calendar



[Bilder, Sound und Animation](#)

Canvas



[Einfache Benutzeroberflächen für Applets](#)

CardLayout



[Benutzerschnittstellen entwerfen](#)

CharArrayReader



[Java und Streams](#)

CharArrayWriter



[Java und Streams](#)

Checkbox

- [Einfache Benutzeroberflächen für Applets](#)

CheckboxGroup

- [Einfache Benutzeroberflächen für Applets](#)



[Benutzerschnittstellen mit Swing](#)

CheckBoxMenuItem

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Choice

- [Einfache Benutzeroberflächen für Applets](#)

- [Einfache Benutzeroberflächen für Applets](#)



[Benutzerschnittstellen mit Swing](#)

Clipboard

- [JavaBeans und andere fortgeschrittene Features](#)

Color

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)



[Ereignisverarbeitung in Applets](#)

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

ColorControls

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

ColorSpace

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

ColorTest

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Component



[Ereignisverarbeitung in Applets](#)

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Constructor

- [Arbeiten mit Objekten](#)

CountInstances

- [Pakete, Schnittstellen und mehr](#)

DataFlavor

- [JavaBeans und andere fortgeschrittene Features](#)

DataInputStream

- [Java und Streams](#)
- [Java und Streams](#)
- [Java und Streams](#)

DataOutputStream

- [Java und Streams](#)

Date

- [Bilder, Sound und Animation](#)

Definieren

- [Java: Eine Klassensprache](#)

Dialog

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Entwurf

- [Objektorientierte Programmierung - ein erster Eindruck](#)

Error

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

erstellen

- [Objektorientierte Programmierung - ein erster Eindruck](#)

Event

- [Ereignisverarbeitung in Applets](#)
-

[Ereignisverarbeitung in Applets](#)



[Ereignisverarbeitung in Applets](#)

EventObject



[Ereignisbehandlung mit Swing](#)

Exception



[Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Field



[Arbeiten mit Objekten](#)

FileDialog



[Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

FileInputStream



[Java und Streams](#)

FileOutputStream



[Java und Streams](#)

FileReader



[Java und Streams](#)



[Java und Streams](#)

FileWriter



[Java und Streams](#)



[Java und Streams](#)

FilterInputStream



[Java und Streams](#)

FilterOutputStream



[Java und Streams](#)

FilterWriter



[Java und Streams](#)

final



[Pakete, Schnittstellen und mehr](#)

FlowLayout



[Benutzerschnittstellen entwerfen](#)



[Benutzerschnittstellen entwerfen](#)

FocusEvent



[Ereignisbehandlung mit Swing](#)

Font



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

FontMetrics



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

Frame



[Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Fruit



[Pakete, Schnittstellen und mehr](#)

GeneralPath



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

getKeyChar()



[Ereignisbehandlung mit Swing](#)

GradientPaint



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

Graphics



[Programme mit Grafik, Fonts und Farbe verfeinern](#)



[Ereignisverarbeitung in Applets](#)

Graphics2D



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

GregorianCalendar

- [Bilder, Sound und Animation](#)

GridBagConstraints

- [Benutzerschnittstellen entwerfen](#)
- [Benutzerschnittstellen entwerfen](#)

GridLayout

- [Benutzerschnittstellen entwerfen](#)

GridLayout

- [Benutzerschnittstellen entwerfen](#)

Icon

- [Benutzerschnittstellen mit Swing](#)

identifizieren

- [Pakete, Schnittstellen und mehr](#)

Image

- [Bilder, Sound und Animation](#)
- [Benutzerschnittstellen mit Swing](#)

InputStream

- [Java und Streams](#)

Insets

- [Benutzerschnittstellen entwerfen](#)

ItemEvent

- [Ereignisbehandlung mit Swing](#)

JButton

- [Benutzerschnittstellen mit Swing](#)
- [Benutzerschnittstellen mit Swing](#)

JCheckBox

- [Benutzerschnittstellen mit Swing](#)
- JComboBox
 - [Benutzerschnittstellen mit Swing](#)
- JComponent
 - [Benutzerschnittstellen mit Swing](#)
- JFrame
 - [Benutzerschnittstellen mit Swing](#)
 - [Ereignisbehandlung mit Swing](#)
- JLabel
 - [Benutzerschnittstellen mit Swing](#)
- JOptionPane
 - [Benutzerschnittstellen mit Swing](#)
 - [Benutzerschnittstellen mit Swing](#)
- JPanel
 - [Benutzerschnittstellen mit Swing](#)
- JPasswordField
 - [Benutzerschnittstellen mit Swing](#)
- JRadioButton
 - [Benutzerschnittstellen mit Swing](#)
- JScrollBar
 - [Benutzerschnittstellen mit Swing](#)
- JTextArea
 - [Benutzerschnittstellen mit Swing](#)
- JTextField
 - [Benutzerschnittstellen mit Swing](#)
- JWindow



[Ereignisbehandlung mit Swing](#)

KeyEvent



[Ereignisbehandlung mit Swing](#)

Label



[Einfache Benutzeroberflächen für Applets](#)

Line2D.Float



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

LinkedList



[Pakete, Schnittstellen und mehr](#)

List



[Einfache Benutzeroberflächen für Applets](#)

Math



[Arbeiten mit Objekten](#)

MenuBar



[Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

MenuItem



[Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Method



[Arbeiten mit Objekten](#)

Modifier



[Arbeiten mit Objekten](#)

MouseEvent



[Ereignisbehandlung mit Swing](#)

Naming



[JavaBeans und andere fortgeschrittene Features](#)

Node



[Pakete, Schnittstellen und mehr](#)

Object



[Objektorientierte Programmierung - ein erster Eindruck](#)



[Java: Eine Klassensprache](#)

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

ObjectInputStream

-

[Java und Streams](#)

ObjectOutputStream

-

[Java und Streams](#)

-

[Java und Streams](#)

Orange

-

[Pakete, Schnittstellen und mehr](#)

organisieren

-

[Pakete, Schnittstellen und mehr](#)

OutputStream

-

[Java und Streams](#)

PipedInputStream

-

[Java und Streams](#)

PipedOutputStream

-

[Java und Streams](#)

PipedReader

-

[Java und Streams](#)

PipedWriter

-

[Java und Streams](#)

Point

-

[Ereignisverarbeitung in Applets](#)

Polygon

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Positionen

- [Pakete, Schnittstellen und mehr](#)

PrintReader

- [Java und Streams](#)

PrintStream

- [Java und Streams](#)

PushbackInputStream

- [Java und Streams](#)

PushbackReader

- [Java und Streams](#)

Random

- [Arbeiten mit Objekten](#)

RandomAccessFile

- [Java und Streams](#)

Reader

- [Java und Streams](#)

Rectangle2D.Double

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Rectangle2D.Float

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Reflection

- [Arbeiten mit Objekten](#)
- [Arbeiten mit Objekten](#)

RuntimeException

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Scrollbar

- [Einfache Benutzeroberflächen für Applets](#)

ServerSocket

- [Kommunikation über das Internet](#)

Socket

- [Kommunikation über das Internet](#)

SocketImpl

- [Kommunikation über das Internet](#)

Sphere

- [Pakete, Schnittstellen und mehr](#)

StreamTokenizer

- [Java und Streams](#)

String

- [Arbeiten mit Objekten](#)

StringSelection

- [JavaBeans und andere fortgeschrittene Features](#)

SwingColorControls

- [Benutzerschnittstellen mit Swing](#)
- [Ereignisbehandlung mit Swing](#)

SwingColorTest

- [Benutzerschnittstellen mit Swing](#)
- [Ereignisbehandlung mit Swing](#)

SwingConstants

- [Benutzerschnittstellen mit Swing](#)

TextArea

- [Einfache Benutzeroberflächen für Applets](#)

TextComponent

- [Einfache Benutzeroberflächen für Applets](#)

TextDialog

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

TextField

- [Einfache Benutzeroberflächen für Applets](#)

Thread

- [Bilder, Sound und Animation](#)

Throwable

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

TriviaServer

- [Kommunikation über das Internet](#)
- [Kommunikation über das Internet](#)

UIManager

- [Benutzerschnittstellen mit Swing](#)

URL

- [Bilder, Sound und Animation](#)
- [Kommunikation über das Internet](#)
- [JavaBeans und andere fortgeschrittene Features](#)

Vector

- [Java und Streams](#)

Verhaltensweisen

- [Objektorientierte Programmierung - ein erster Eindruck](#)

Window

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Writer

- [Java und Streams](#)

Klassenbibliothek

- [Objektorientierte Programmierung - ein erster Eindruck](#)

Klassenhierarchie

- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Pakete, Schnittstellen und mehr erzeugen](#)
- [Objektorientierte Programmierung - ein erster Eindruck](#)

Klassenmethode

- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Arbeiten mit Objekten](#)
- [Java: Eine Klassensprache](#)
- [Pakete, Schnittstellen und mehr](#)

Klassentypen

- [Das Java-ABC](#)

Klassenvariable

- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Das Java-ABC](#)
- [Arbeiten mit Objekten](#)
- [Java: Eine Klassensprache](#)
- [Pakete, Schnittstellen und mehr](#)

Kleiner, Perkins, Caufield and Byers

- [Eine Handvoll Java](#)

Kommandozeile

- [Die Konfiguration des Java Development Kit](#)

Kommentar

- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Das Java-ABC](#)

Komponente

Bildlaufleisten

- [Einfache Benutzeroberflächen für Applets](#)

Canvas

- [Einfache Benutzeroberflächen für Applets](#)

Dialogfelder

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)
- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Fenster

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Frames

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Kombinationslistenfelder

- [Einfache Benutzeroberflächen für Applets](#)

Kontrollfelder

- [Einfache Benutzeroberflächen für Applets](#)

Labels

- [Einfache Benutzeroberflächen für Applets](#)

Listenfelder

- [Einfache Benutzeroberflächen für Applets](#)

Mehrzeilige Textfelder

- [Einfache Benutzeroberflächen für Applets](#)

Menleisten

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Mens

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

nicht visuelle

- [JavaBeans und andere fortgeschrittene Features](#)

Textfelder

- [Einfache Benutzeroberflächen für Applets](#)

Konstanten, Definition

- [Java: Eine Classesprache](#)

Konstruktor

- [Arbeiten mit Objekten](#)

anderen aufrufen

- [Mehr über Methoden](#)

berladen



- [Mehr über Methoden](#)

berschreiben



- [Mehr über Methoden](#)

Definition



- [Arbeiten mit Objekten](#)

Konvention, Paketnamen



- [Pakete, Schnittstellen und mehr](#)

Koordinatensystem



- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

KPCB



- [Eine Handvoll Java](#)

Krzysztof Kieslowski



- [Einfache Benutzeroberflächen für Applets](#)

L

Label-Klasse



- [Einfache Benutzeroberflächen für Applets](#)

Layout-Manager



- [Benutzerschnittstellen entwerfen](#)



- [Benutzerschnittstellen mit Swing](#)

BorderLayout



- [Benutzerschnittstellen entwerfen](#)



- [Benutzerschnittstellen entwerfen](#)



- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

CardLayout



- [Benutzerschnittstellen entwerfen](#)



- [Benutzerschnittstellen entwerfen](#)

FlowLayout

- [Benutzerschnittstellen entwerfen](#)

GridBagLayout

- [Benutzerschnittstellen entwerfen](#)

- [Benutzerschnittstellen entwerfen](#)

GridLayout

- [Benutzerschnittstellen entwerfen](#)

- [Benutzerschnittstellen entwerfen](#)

verschiedene gleichzeitig

- [Benutzerschnittstellen entwerfen](#)

leakOil()-Methode

- [Pakete, Schnittstellen und mehr](#)

LEGO-Steine

- [Objektorientierte Programmierung - ein erster Eindruck](#)

Lines-Applet

- [Ereignisverarbeitung in Applets](#)

Linien

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

LinkedList-Klasse

- [Pakete, Schnittstellen und mehr](#)

Listener

- [Ereignisbehandlung mit Swing](#)

Listings

GetRaven-Klasse

- [Kommunikation über das Internet](#)

Reflection

- [Arbeiten mit Objekten](#)

List-Klasse

- [Einfache Benutzeroberflächen für Applets](#)

Literal



- [Das Java-ABC](#)

- boolesche



- [Das Java-ABC](#)

- String



- [Das Java-ABC](#)

- Zahlen



- [Das Java-ABC](#)

- Zeichen



- [Das Java-ABC](#)

Lokale Variablen



- [Das Java-ABC](#)

Look and Feel



- [Benutzerschnittstellen mit Swing](#)



- [Benutzerschnittstellen mit Swing](#)



- [Benutzerschnittstellen mit Swing](#)

lostFocus()-Methode



- [Ereignisverarbeitung in Applets](#)

lostOwnership()-Methode



- [JavaBeans und andere fortgeschrittene Features](#)

M

Macromedia Dreamweaver



- [Objektorientierte Programmierung - ein erster Eindruck](#)



[Grundlagen der Java-Applets](#)

main()-Methode

- [Objektorientierte Programmierung - ein erster Eindruck](#)

Marc Andreesen

- [Eine Handvoll Java](#)

mark()-Methode

-

[Java und Streams](#)

markSupported()-Methoden

-

[Java und Streams](#)

Matryoska

-

[Das Java-ABC](#)

Mausereignisse

- [Ereignisbehandlung mit Swing](#)

Maustastenkombinationen

- [Ereignisverarbeitung in Applets](#)

Mehrfachvererbung

- [Objektorientierte Programmierung - ein erster Eindruck](#)

Meldungen, Anzeigen Statuszeile

- [JavaBeans und andere fortgeschrittene Features](#)

Menereignisse

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

MenuBar-Klasse

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

MenuItem-Klasse

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Meta

- [Ereignisverarbeitung in Applets](#)

metaDown()-Methode

- [Ereignisverarbeitung in Applets](#)

Methode

- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Objektorientierte Programmierung - ein erster Eindruck](#)

abstract



[Pakete, Schnittstellen und mehr](#)

Accessor



[Pakete, Schnittstellen und mehr](#)

action()



[Ereignisverarbeitung in Applets](#)



[Ereignisverarbeitung in Applets](#)



[Fortgeschrittene Benutzeroberflächen mit dem AWT](#)



[Ereignisbehandlung mit Swing](#)

actionPerformed()



[Ereignisbehandlung mit Swing](#)

add()



[Einfache Benutzeroberflächen für Applets](#)



[Einfache Benutzeroberflächen für Applets](#)



[Fortgeschrittene Benutzeroberflächen mit dem AWT](#)



[Benutzerschnittstellen mit Swing](#)



[Benutzerschnittstellen mit Swing](#)

addActionListener()



[Ereignisbehandlung mit Swing](#)



[Ereignisbehandlung mit Swing](#)

addAdjustmentListener()



[Ereignisbehandlung mit Swing](#)

addFocusListener()



[Ereignisbehandlung mit Swing](#)

addInstance()

- [Pakete, Schnittstellen und mehr](#)
- addItem()
 - [Einfache Benutzeroberflächen für Applets](#)
 - [Benutzerschnittstellen mit Swing](#)
- addItemListener()
 - [Ereignisbehandlung mit Swing](#)
- addKeyListener()
 - [Ereignisbehandlung mit Swing](#)
- addline()
 - [Ereignisverarbeitung in Applets](#)
- addMotionListener()
 - [Ereignisbehandlung mit Swing](#)
- addMouseListener()
 - [Ereignisbehandlung mit Swing](#)
- addspot()
 - [Ereignisverarbeitung in Applets](#)
- addWindowListener()
 - [Ereignisbehandlung mit Swing](#)
- adjustmentValueChanged()
 - [Ereignisbehandlung mit Swing](#)
- Argumentenliste
 - [Java: Eine Klassensprache](#)
- Aufrufen
 - [Arbeiten mit Objekten](#)
- available()
 - [Java und Streams](#)

berladung

- [Mehr über Methoden](#)

beschreiben

- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Mehr über Methoden](#)

buildConstraints

- [Benutzerschnittstellen entwerfen](#)

changeColor()

- [Ereignisverarbeitung in Applets](#)

charAt()

- [Arbeiten mit Objekten](#)

charWidth()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

clearRect()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

close()

- [Java und Streams](#)

- [Java und Streams](#)

- [Java und Streams](#)

closePath()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

controlDown()

- [Ereignisverarbeitung in Applets](#)

copyArea()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

countItems()

- [Einfache Benutzeroberflächen für Applets](#)
- [Einfache Benutzeroberflächen für Applets](#)

Definition



[Java: Eine Klassesprache](#)

destroy



[Grundlagen der Java-Applets](#)

destroy()



[Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

disable()



[Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

dispose()



[Bilder, Sound und Animation](#)

draw()



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

drawArc()



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

drawImage()



[Bilder, Sound und Animation](#)

drawLine()



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

drawOval()



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

drawPolygon()



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

drawPolyline()



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

drawRect()



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

drawRoundedRect()



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

drawString()



[Programme mit Grafik, Fonts und Farbe verfeinern](#)



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

electricalSystemFailure()



[Pakete, Schnittstellen und mehr](#)

elementAt()



[Java und Streams](#)

enable()

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

fill()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

fillArc()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

fillOval()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

fillPolygon()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

fillRect()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

fillRoundedRect()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

final



[Pakete, Schnittstellen und mehr](#)

Finalize()



[Mehr über Methoden](#)

finalize()



[Java und Streams](#)

flush()



[Java und Streams](#)



[Java und Streams](#)



[Java und Streams](#)

focusGained()



[Ereignisbehandlung mit Swing](#)

focusLost()



[Ereignisbehandlung mit Swing](#)

Fr ByteArrayOutputStream



[Java und Streams](#)

Fr CharArrayWriter



[Java und Streams](#)

Fr DataInput



[Java und Streams](#)

Fr PrintStream



[Java und Streams](#)

getAccessibleContext()



[Benutzerschnittstellen mit Swing](#)

getActionCommand()



[Ereignisbehandlung mit Swing](#)

getAdjustmentType()



[Ereignisbehandlung mit Swing](#)

getAppletContext()



[Kommunikation über das Internet](#)

■ [JavaBeans und andere fortgeschrittene Features](#)

■ [JavaBeans und andere fortgeschrittene Features](#)

getApplets()

■ [JavaBeans und andere fortgeschrittene Features](#)

getAudioClip()



[Bilder, Sound und Animation](#)

getBackground()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

getBuffer()

-

[Java und Streams](#)

getClickCount()

-

[Ereignisbehandlung mit Swing](#)

getCodeBase()

-

[Bilder, Sound und Animation](#)

getColor()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

getContext

- [JavaBeans und andere fortgeschrittene Features](#)

getCrossPlatformLookAndFeelClassName()

-

[Benutzerschnittstellen mit Swing](#)

getCurrent()

- [Einfache Benutzeroberflächen für Applets](#)

getDocumentBase()

-

[Bilder, Sound und Animation](#)

getFD()

-

[Java und Streams](#)

-

[Java und Streams](#)

getFontMetrics()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

getForeground()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

getHeight()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

getHSBColor()

-

[Bilder, Sound und Animation](#)

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

getImage()



- [Bilder, Sound und Animation](#)

getInputStream()



- [Kommunikation über das Internet](#)

getInsets()



- [Benutzerschnittstellen entwerfen](#)



- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)



- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)



- [Benutzerschnittstellen mit Swing](#)

getItem()



- [Einfache Benutzeroberflächen für Applets](#)



- [Einfache Benutzeroberflächen für Applets](#)



- [Ereignisbehandlung mit Swing](#)

getItemCount()



- [Einfache Benutzeroberflächen für Applets](#)

getKeyChar()



- [Ereignisbehandlung mit Swing](#)

getLabel()



- [Einfache Benutzeroberflächen für Applets](#)



- [Ereignisverarbeitung in Applets](#)

getNumInstances()



- [Pakete, Schnittstellen und mehr](#)

getParent()



- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

getPoint()



- [Ereignisbehandlung mit Swing](#)

getSelectedIndex()



- [Einfache Benutzeroberflächen für Applets](#)



- [Einfache Benutzeroberflächen für Applets](#)

getSelectedItem()

- [Einfache Benutzeroberflächen für Applets](#)

- [Einfache Benutzeroberflächen für Applets](#)

getSize()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

getSource()

- [Ereignisbehandlung mit Swing](#)

getState()

- [Einfache Benutzeroberflächen für Applets](#)

getStateChange()

- [Ereignisbehandlung mit Swing](#)

getSystemClipboard()

- [JavaBeans und andere fortgeschrittene Features](#)

getSystemLookAndFeelClassName()

- [Benutzerschnittstellen mit Swing](#)

getText()

- [Einfache Benutzeroberflächen für Applets](#)

- [Einfache Benutzeroberflächen für Applets](#)

- [Einfache Benutzeroberflächen für Applets](#)

getToolkit()

- [JavaBeans und andere fortgeschrittene Features](#)

getTransferData()

- [JavaBeans und andere fortgeschrittene Features](#)

- [JavaBeans und andere fortgeschrittene Features](#)

getTransferDataFlavors()

- [JavaBeans und andere fortgeschrittene Features](#)

- [JavaBeans und andere fortgeschrittene Features](#)

getValue()

- [Einfache Benutzeroberflächen für Applets](#)

- [Ereignisbehandlung mit Swing](#)

getY()

- [Ereignisbehandlung mit Swing](#)

gotFocus()

-

[Ereignisverarbeitung in Applets](#)

handleChoice()



[Ereignisverarbeitung in Applets](#)

handleEvent()



[Ereignisverarbeitung in Applets](#)



[Ereignisverarbeitung in Applets](#)



[Ereignisverarbeitung in Applets](#)

■ [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

■ [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

handleText()



[Ereignisverarbeitung in Applets](#)

hide()

■ [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

indexOf()



[Arbeiten mit Objekten](#)

initQnA()



[Kommunikation über das Internet](#)

insert()

■ [Einfache Benutzeroberflächen für Applets](#)



[Pakete, Schnittstellen und mehr](#)

insertObject()



[Pakete, Schnittstellen und mehr](#)

insertText()

■ [Einfache Benutzeroberflächen für Applets](#)

insets()



[Benutzerschnittstellen entwerfen](#)

isDataFlavorSupported()

■ [JavaBeans und andere fortgeschrittene Features](#)

isEditable()

- [Einfache Benutzeroberflächen für Applets](#)

- [Einfache Benutzeroberflächen für Applets](#)

itemStateChanged()

- [Ereignisbehandlung mit Swing](#)

keyDown()

- [Ereignisverarbeitung in Applets](#)

- [Ereignisbehandlung mit Swing](#)

keyPressed()

- [Ereignisbehandlung mit Swing](#)

keyReleased()

- [Ereignisbehandlung mit Swing](#)

keyTyped()

- [Ereignisbehandlung mit Swing](#)

keyUp()

- [Ereignisverarbeitung in Applets](#)

Konstruktor

- [Mehr über Methoden](#)

leakOil()

- [Pakete, Schnittstellen und mehr](#)

loop()

- [Bilder, Sound und Animation](#)

lostFocus()

- [Ereignisverarbeitung in Applets](#)

lostOwnership()

- [JavaBeans und andere fortgeschrittene Features](#)

main()

- [Java: Eine Klassensprache](#)

mark()



[Java und Streams](#)

markSupported()



[Java und Streams](#)

metaDown()



[Ereignisverarbeitung in Applets](#)

mouseClicked()



[Ereignisbehandlung mit Swing](#)

mouseDown()



[Ereignisverarbeitung in Applets](#)



[Ereignisverarbeitung in Applets](#)



[Ereignisbehandlung mit Swing](#)

mouseDrag()



[Ereignisverarbeitung in Applets](#)

mouseDragged()



[Ereignisbehandlung mit Swing](#)

mouseEnter()



[Ereignisverarbeitung in Applets](#)

mouseEntered()



[Ereignisbehandlung mit Swing](#)

mouseExit()



[Ereignisverarbeitung in Applets](#)

mouseExited()



[Ereignisbehandlung mit Swing](#)

mouseMove()



[Ereignisverarbeitung in Applets](#)

mouseMoved()

- [Ereignisbehandlung mit Swing](#)

mousePressed()

- [Ereignisbehandlung mit Swing](#)

mouseReleased()

- [Ereignisbehandlung mit Swing](#)

mouseUp()

- [Ereignisverarbeitung in Applets](#)

moveTo()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

newAudioClip()

- [Bilder, Sound und Animation](#)

openStream()

- [Kommunikation über das Internet](#)
- [Kommunikation über das Internet](#)

original Methode aufrufen

- [Mehr über Methoden](#)

pack()

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)
- [Benutzerschnittstellen mit Swing](#)

paint()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)
- [Bilder, Sound und Animation](#)

Parameter

- [Mehr über Methoden](#)

Parameter bergeben

- [Java: Eine Klassesprache](#)

play()



[Bilder, Sound und Animation](#)

processInput()



[Kommunikation über das Internet](#)

Rückgabotyp



[Java: Eine Klassensprache](#)

read()



[Java und Streams](#)

readLine()



[Java und Streams](#)



[Java und Streams](#)

readObject()



[Java und Streams](#)

readUTF()



[Java und Streams](#)



[Java und Streams](#)

ready()



[Java und Streams](#)

Reflection



[Arbeiten mit Objekten](#)



[Arbeiten mit Objekten](#)

repaint()



[Bilder, Sound und Animation](#)

replace()

- [Einfache Benutzeroberflächen für Applets](#)

replaceText()

- [Einfache Benutzeroberflächen für Applets](#)

requestFocus()

- [Ereignisverarbeitung in Applets](#)

reset()

- [Java und Streams](#)

- [Java und Streams](#)

reshape()

- [Benutzerschnittstellen entwerfen](#)

resize()

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

RGBtoHSB()

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

run()

- [Bilder, Sound und Animation](#)

select()

- [Einfache Benutzeroberflächen für Applets](#)
- [Einfache Benutzeroberflächen für Applets](#)

sendMessage()

- [JavaBeans und andere fortgeschrittene Features](#)

setAccessibleDescription()

- [Benutzerschnittstellen mit Swing](#)

setAccessibleName()

- [Benutzerschnittstellen mit Swing](#)

setActionCommand()

- [Ereignisbehandlung mit Swing](#)

setBackground()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)
- setColor()
- [Programme mit Grafik, Fonts und Farbe verfeinern](#)
- [Programme mit Grafik, Fonts und Farbe verfeinern](#)
- setConstraints()
- - [Benutzerschnittstellen entwerfen](#)
- setContentPane()
- - [Benutzerschnittstellen mit Swing](#)
- setContents()
- [JavaBeans und andere fortgeschrittene Features](#)
- setCurrent()
- [Einfache Benutzeroberflächen für Applets](#)
- setEchoChar()
- [Einfache Benutzeroberflächen für Applets](#)
- - [Benutzerschnittstellen mit Swing](#)
- setEditable()
- [Einfache Benutzeroberflächen für Applets](#)
- [Einfache Benutzeroberflächen für Applets](#)
- - [Benutzerschnittstellen mit Swing](#)
- setFont()
- [Einfache Benutzeroberflächen für Applets](#)
- setFont()
- [Programme mit Grafik, Fonts und Farbe verfeinern](#)
- setForeground()
- [Programme mit Grafik, Fonts und Farbe verfeinern](#)
- setHelpMenu()
- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)
- setLabel()
- [Einfache Benutzeroberflächen für Applets](#)
- setLayout()
- - [Benutzerschnittstellen entwerfen](#)
- - [Benutzerschnittstellen mit Swing](#)

setLookAndFeel()



- [Benutzerschnittstellen mit Swing](#)

setMenuBar()

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

setMnemonic()



- [Benutzerschnittstellen mit Swing](#)

setPaint()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

setState()

- [Einfache Benutzeroberflächen für Applets](#)

setStroke()

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

setText()

- [Einfache Benutzeroberflächen für Applets](#)

- [Einfache Benutzeroberflächen für Applets](#)

- [Einfache Benutzeroberflächen für Applets](#)



- [Ereignisbehandlung mit Swing](#)

setToolTipText()



- [Benutzerschnittstellen mit Swing](#)

setValue()

- [Einfache Benutzeroberflächen für Applets](#)

setVisible()



- [Benutzerschnittstellen mit Swing](#)

shiftDown()



- [Ereignisverarbeitung in Applets](#)

showConfirmDialog()



- [Benutzerschnittstellen mit Swing](#)

showDocument()



- [Kommunikation über das Internet](#)

- [JavaBeans und andere fortgeschrittene Features](#)

showInputDialog()

- [Benutzerschnittstellen mit Swing](#)
showMessageDialog()
- [Benutzerschnittstellen mit Swing](#)
showOptionDialog()
- [Benutzerschnittstellen mit Swing](#)
showStatus()
- [JavaBeans und andere fortgeschrittene Features](#)
Signatur
- [Java: Eine Klassesprache](#)
size()
- [Programme mit Grafik, Fonts und Farbe verfeinern](#)
- [Pakete, Schnittstellen und mehr](#)
skip()
- [Java und Streams](#)
skipBytes()
- [Java und Streams](#)
sleep()
- [Bilder, Sound und Animation](#)
start()
- [Grundlagen der Java-Applets](#)
stop()
- [Grundlagen der Java-Applets](#)
stringWidth()
- [Programme mit Grafik, Fonts und Farbe verfeinern](#)
Substring()
- [Arbeiten mit Objekten](#)
theList()

- [Pakete, Schnittstellen und mehr](#)
- this()
- [Mehr über Methoden](#)
- toCharArray()
- [Java und Streams](#)
- toString()
- [Bilder, Sound und Animation](#)
- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)
- [Ereignisbehandlung mit Swing](#)
- toUpperCase()
- [Arbeiten mit Objekten](#)
- unread()
- [Java und Streams](#)
- [Java und Streams](#)
- update()
- [Bilder, Sound und Animation](#)
- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)
- valueOf()
- [Arbeiten mit Objekten](#)
- Verschachteln
- [Arbeiten mit Objekten](#)
- windowActivated()
- [Ereignisbehandlung mit Swing](#)
- windowClosed()
- [Ereignisbehandlung mit Swing](#)

windowClosing()



- [Ereignisbehandlung mit Swing](#)

windowDeactivated()



- [Ereignisbehandlung mit Swing](#)

windowDeiconified()



- [Ereignisbehandlung mit Swing](#)

windowIconified()



- [Ereignisbehandlung mit Swing](#)

windowOpened()



- [Ereignisbehandlung mit Swing](#)

write()



- [Java und Streams](#)



- [Java und Streams](#)

writeTo()



- [Java und Streams](#)

Zugriffskontrolle



- [Pakete, Schnittstellen und mehr](#)

Methodendefinitionen



- [Pakete, Schnittstellen und mehr](#)

Method-Klasse



- [Arbeiten mit Objekten](#)

Microsoft Frontpage



- [Objektorientierte Programmierung - ein erster Eindruck](#)



- [Grundlagen der Java-Applets](#)

Microsoft Internet Explorer



[Grundlagen der Java-Applets](#)

Microsoft Java Software Development Kit

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

MIDI-Formate

RMF



[Bilder, Sound und Animation](#)

Typ 0 MIDI



[Bilder, Sound und Animation](#)

MIME-Typen

Datenbertragungsobjekte



[JavaBeans und andere fortgeschrittene Features](#)

Web-Standort



[JavaBeans und andere fortgeschrittene Features](#)

Mnemonics



[Benutzerschnittstellen mit Swing](#)



[Benutzerschnittstellen mit Swing](#)

Modale Dialogfelder

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Modifier



[Pakete, Schnittstellen und mehr](#)

abstract



[Pakete, Schnittstellen und mehr](#)



[Pakete, Schnittstellen und mehr](#)

final



[Pakete, Schnittstellen und mehr](#)



[Pakete, Schnittstellen und mehr](#)

private



[Pakete, Schnittstellen und mehr](#)

protected



[Pakete, Schnittstellen und mehr](#)

public



[Pakete, Schnittstellen und mehr](#)

static



[Pakete, Schnittstellen und mehr](#)

synchronized



[Pakete, Schnittstellen und mehr](#)

volatile



[Pakete, Schnittstellen und mehr](#)

Modifier-Klasse



[Arbeiten mit Objekten](#)

Modulo-Operation



[Das Java-ABC](#)

mouseClicked()-Methode



[Ereignisbehandlung mit Swing](#)

mouseDown()-Methode



[Ereignisverarbeitung in Applets](#)



[Ereignisverarbeitung in Applets](#)



[Ereignisbehandlung mit Swing](#)

mouseDown-Ereignisse



[Ereignisverarbeitung in Applets](#)

mouseDrag()-Methode



[Ereignisverarbeitung in Applets](#)

mouseDragged()-Methode



[Ereignisbehandlung mit Swing](#)

mouseEnter()-Methode



[Ereignisverarbeitung in Applets](#)

mouseEntered()-Methode



[Ereignisbehandlung mit Swing](#)

MouseEvent-Klasse



[Ereignisbehandlung mit Swing](#)

mouseExit()-Methode



[Ereignisverarbeitung in Applets](#)

mouseExited()-Methode



[Ereignisbehandlung mit Swing](#)

MouseListener-Schnittstelle



[Ereignisbehandlung mit Swing](#)

MouseMotion-Ereignisse



[Ereignisbehandlung mit Swing](#)

MouseMotionListener-Schnittstelle



[Ereignisbehandlung mit Swing](#)

mouseMove()-Methode



[Ereignisverarbeitung in Applets](#)

mouseMoved()-Methode



[Ereignisbehandlung mit Swing](#)

mousePressed()-Methode



[Ereignisbehandlung mit Swing](#)

mouseReleased()-Methode



[Ereignisbehandlung mit Swing](#)

mouseUp()-Methode



[Ereignisverarbeitung in Applets](#)

MS-DOS-Eingabeaufforderung



[Die Konfiguration des Java Development Kit](#)

MS-DOS-Prompt

- [Die Konfiguration des Java Development Kit](#)

Multitasking

- [Bilder, Sound und Animation](#)

N

Namen für Pakete und Klassen

- [Pakete, Schnittstellen und mehr](#)

Namenskonflikt

- [Pakete, Schnittstellen und mehr](#)

Naming-Klasse

- [JavaBeans und andere fortgeschrittene Features](#)

Natural Intelligence Roaster

- [Eine Handvoll Java](#)

Navigator

- [Grundlagen der Java-Applets](#)

Neil/Fred's Gigantic List of Palindromes

- [Objektorientierte Programmierung - ein erster Eindruck](#)

Neko-Applet

- [Bilder, Sound und Animation](#)

Netscape

- [Eine Handvoll Java](#)

Netscape Navigator

- [Grundlagen der Java-Applets](#)

Netscape Object Signing Software Publishing ID

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Netscape Signing-Tool

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Newsgroup

- [Eine Handvoll Java](#)

Node-Klasse

- [Pakete, Schnittstellen und mehr](#)

Notepad

- [Texteditoren und das JDK](#)

O

Oak

- [Eine Handvoll Java](#)

ObjectInputStream-Klasse

- [Java und Streams](#)

Object-Klasse

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

ObjectOutputStream-Klasse

- [Java und Streams](#)
- [Java und Streams](#)

OBJECT-Tag

- [Grundlagen der Java-Applets](#)

Objekt

- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Objektorientierte Programmierung - ein erster Eindruck](#)
Datenbertragung

Konvertieren

- [Arbeiten mit Objekten](#)

Point

- [Arbeiten mit Objekten](#)

Referenzen auf

- [Arbeiten mit Objekten](#)

Vergleichen



[Arbeiten mit Objekten](#)

Wrapper



[Arbeiten mit Objekten](#)

Objektorientierte Programmierung



[Eine Handvoll Java](#)



[Objektorientierte Programmierung - ein erster Eindruck](#)

Objektvariable



[Objektorientierte Programmierung - ein erster Eindruck](#)

Objekt-Wrapper



[Arbeiten mit Objekten](#)

OCX-Komponentenarchitekturen



[JavaBeans und andere fortgeschrittene Features](#)

Oktal-System



[Das Java-ABC](#)

OOP



[Eine Handvoll Java](#)



[Objektorientierte Programmierung - ein erster Eindruck](#)



[Pakete, Schnittstellen und mehr](#)

openStream()-Methode



[Kommunikation über das Internet](#)



[Kommunikation über das Internet](#)

Operationen der Zwischenablage

Datenbertragungsobjekte



[JavaBeans und andere fortgeschrittene Features](#)



[JavaBeans und andere fortgeschrittene Features](#)

Operator



[Das Java-ABC](#)

arithmetische



[Das Java-ABC](#)

Bedingung



[Arrays, Bedingungen und Schleifen](#)

inkrement/dekrement



[Das Java-ABC](#)

InstanceOf



[Arbeiten mit Objekten](#)

logische



[Das Java-ABC](#)

New



[Arbeiten mit Objekten](#)

Prefix/Postfix



[Das Java-ABC](#)

Przedenz



[Das Java-ABC](#)

ternrer



[Arrays, Bedingungen und Schleifen](#)

vergleich



[Das Java-ABC](#)

Zuweisung



[Das Java-ABC](#)

Orange-Klasse



[Pakete, Schnittstellen und mehr](#)

Organisation von Klassen

- [Pakete, Schnittstellen und mehr](#)

OutputStream-Klasse

- [Java und Streams](#)

Ovale

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

P

pack()-Methode

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)
- [Benutzerschnittstellen mit Swing](#)

package-Anweisung

- [Pakete, Schnittstellen und mehr](#)

Packages

- [Objektorientierte Programmierung - ein erster Eindruck](#)

Paket

- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Pakete, Schnittstellen und mehr](#)

eigene erstellen

- [Pakete, Schnittstellen und mehr](#)

java

- [Objektorientierte Programmierung - ein erster Eindruck](#)

java.Applet

- [Grundlagen der Java-Applets](#)

java.applet

- [Objektorientierte Programmierung - ein erster Eindruck](#)

java.awt

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)
- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

java.awt.geom

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Java.lang

- [Arbeiten mit Objekten](#)

java.lang

- [Objektorientierte Programmierung - ein erster Eindruck](#)

- [Bilder, Sound und Animation](#)

java.util

- [Arbeiten mit Objekten](#)

Namenskonvention

- [Pakete, Schnittstellen und mehr](#)

Verzeichnisstruktur

- [Pakete, Schnittstellen und mehr](#)

Paket-Hierarchie

- [Pakete, Schnittstellen und mehr](#)

Paketnamen

- [Pakete, Schnittstellen und mehr](#)

Palindrom

- [Objektorientierte Programmierung - ein erster Eindruck](#)

Panels aktualisieren

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Parameter an Applets weitergeben

- [Grundlagen der Java-Applets](#)

PARAM-Tag

- [Grundlagen der Java-Applets](#)
- [Grundlagen der Java-Applets](#)

Pardon My Icons!

- [Benutzerschnittstellen mit Swing](#)

PATH-Einstellungen

berprüfen

- [Die Konfiguration des Java Development Kit](#)

JDK-Ordner nicht enthalten

- [Die Konfiguration des Java Development Kit](#)

ndern

- [Die Konfiguration des Java Development Kit](#)

nicht vorhanden

- [Die Konfiguration des Java Development Kit](#)

Patrick Naughton

-

- [Eine Handvoll Java](#)

Paulina Porizkova

- [Einfache Benutzeroberflächen für Applets](#)

Peer-Klassen

-

- [Benutzerschnittstellen entwerfen](#)

Peers

-

- [Benutzerschnittstellen entwerfen](#)

Persistenz

- [JavaBeans und andere fortgeschrittene Features](#)

Personal Bookshelf

- [Einfache Benutzeroberflächen für Applets](#)

Pipe

-

- [Java und Streams](#)

PipedInputStream-Klasse

-

- [Java und Streams](#)

PipedOutputStream-Klasse

-

- [Java und Streams](#)

PipedReader-Klasse

-

- [Java und Streams](#)

PipedWriter-Klasse

-

[Java und Streams](#)

Plattformabhngigkeit

- [Eine Handvoll Java](#)

Plattformunabhngigkeit

- [Eine Handvoll Java](#)

Pointer

- [Arbeiten mit Objekten](#)

Point-Klasse

- [Ereignisverarbeitung in Applets](#)

policytool

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Polygone

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Ports, Sockets

- [Kommunikation über das Internet](#)

Primitivtypen

- [Arbeiten mit Objekten](#)

PrintReader-Klasse

- [Java und Streams](#)

PrintStream-Klasse

- [Java und Streams](#)

Privater Schlssel

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

processInput()-Methode

- [Kommunikation über das Internet](#)

Programmierung, Datenstrme

-

[Java und Streams](#)

Pseudo-Zufallszahlen

- [Arbeiten mit Objekten](#)

public-Klassen

- [Pakete, Schnittstellen und mehr](#)

Pufferbereich einlesen

-

[Java und Streams](#)

Punkt-Notation

- [Arbeiten mit Objekten](#)

Punkt-Operator

- [Arbeiten mit Objekten](#)

PushbackInputStream-Klasse

-

[Java und Streams](#)

PushbackReader-Klasse

-

[Java und Streams](#)

R

RandomAccessFile-Klasse

-

[Java und Streams](#)

Raumschiff Enterprise

- [Eine Handvoll Java](#)

Rckgabetyt

- [Java: Eine Klassensprache](#)

Rckgabewert

-

[Das Java-ABC](#)

read()-Methode

-

[Java und Streams](#)

Reader-Klasse

-

[Java und Streams](#)

readLine()-Methode

-

[Java und Streams](#)

readObject()-Methoden

-

[Java und Streams](#)

readUTF()-Methode

-

[Java und Streams](#)

-

[Java und Streams](#)

ready()-Methode

-

[Java und Streams](#)

Rechtecke

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Reflection

-

[Arbeiten mit Objekten](#)

-

[Arbeiten mit Objekten](#)

Listing



[Arbeiten mit Objekten](#)

Relationale Datenbanken

- [JavaBeans und andere fortgeschrittene Features](#)

Remote Method Invocation (RMI)

- [JavaBeans und andere fortgeschrittene Features](#)

Remote Reference Layer

- [JavaBeans und andere fortgeschrittene Features](#)

Remote-Objekt

- [JavaBeans und andere fortgeschrittene Features](#)

replace()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

replaceText()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

requestFocus()-Methode

- [Ereignisverarbeitung in Applets](#)

reset()-Methode

- [Java und Streams](#)
- [Java und Streams](#)

reshape()-Methode

- [Benutzerschnittstellen entwerfen](#)

resize()-Methode

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

RGBtoHSB()-Methode

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

RMI

Anwendung

Ebenen

- [JavaBeans und andere fortgeschrittene Features](#)

Eigenschaften

- [JavaBeans und andere fortgeschrittene Features](#)
- [JavaBeans und andere fortgeschrittene Features](#)

Erstellen

- [JavaBeans und andere fortgeschrittene Features](#)

im Vergleich zu RPC

- [JavaBeans und andere fortgeschrittene Features](#)

Methodenargumente

- [JavaBeans und andere fortgeschrittene Features](#)

RMI (Remote Method Invocation)

- [JavaBeans und andere fortgeschrittene Features](#)

Rogue Wave JFactory

- [Eine Handvoll Java](#)

RPC im Vergleich zu RMI

- [JavaBeans und andere fortgeschrittene Features](#)

Runnable

- [Pakete, Schnittstellen und mehr](#)

RuntimeException-Klasse

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

S

Schleifen

benennen

- [Arrays, Bedingungen und Schleifen](#)

do

- [Arrays, Bedingungen und Schleifen](#)

for

- [Arrays, Bedingungen und Schleifen](#)

unterbrechen

- [Arrays, Bedingungen und Schleifen](#)

while

- [Arrays, Bedingungen und Schleifen](#)

Schlssel

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Schlsselwort

class

- [Java: Eine Klassensprache](#)

extends

- [Java: Eine Klassensprache](#)

- [Bilder, Sound und Animation](#)

implements

- [Bilder, Sound und Animation](#)

public

- [Objektorientierte Programmierung - ein erster Eindruck](#)

return

- [Java: Eine Klassesprache](#)

static

- [Java: Eine Klassesprache](#)

Super

- [Mehr über Methoden](#)

this

- [Java: Eine Klassesprache](#)

Schnittstelle

- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Pakete, Schnittstellen und mehr](#)

andere Verwendungen

- [Pakete, Schnittstellen und mehr](#)

DataInput

- [Java und Streams](#)

DataOutput

- [Java und Streams](#)

Externalizable

- [Java und Streams](#)

ImageObserver

- [Bilder, Sound und Animation](#)

mehrere implementieren



[Pakete, Schnittstellen und mehr](#)

neue



[Pakete, Schnittstellen und mehr](#)

ObjectInput



[Java und Streams](#)

ObjectOutput



[Java und Streams](#)

Runnable



[Bilder, Sound und Animation](#)

Serializable



[Java und Streams](#)

Schriften verwenden

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Schutzebenen

private



[Pakete, Schnittstellen und mehr](#)

protected



[Pakete, Schnittstellen und mehr](#)

public



[Pakete, Schnittstellen und mehr](#)

standard



[Pakete, Schnittstellen und mehr](#)

Scrollbar-Klasse

- [Einfache Benutzeroberflächen für Applets](#)

select()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

- [Einfache Benutzeroberflächen für Applets](#)

sendMessage()-Methode

- [JavaBeans und andere fortgeschrittene Features](#)

Serialisation

-

[Java und Streams](#)

-

[Java und Streams](#)

- [JavaBeans und andere fortgeschrittene Features](#)
- [JavaBeans und andere fortgeschrittene Features](#)

Serializeable-Schnittstelle

- [JavaBeans und andere fortgeschrittene Features](#)

Serverseitige Sockets

-

[Kommunikation über das Internet](#)

ServerSocket-Klasse

-

[Kommunikation über das Internet](#)

setAccessibleDescription()-Methode

-

[Benutzerschnittstellen mit Swing](#)

setAccessibleName()-Methode

-

[Benutzerschnittstellen mit Swing](#)

setActionCommand()-Methode

-

[Ereignisbehandlung mit Swing](#)

setConstraints()-Method

-

[Benutzerschnittstellen entwerfen](#)

setContentPane()-Methode

-

[Benutzerschnittstellen mit Swing](#)

setContents()-Methode

- [JavaBeans und andere fortgeschrittene Features](#)

setCurrent()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

setEchoChar()-Methode



[Benutzerschnittstellen mit Swing](#)

setEchoCharacter()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

setEditable()-Methode

- [Einfache Benutzeroberflächen für Applets](#)
- [Einfache Benutzeroberflächen für Applets](#)
-

[Benutzerschnittstellen mit Swing](#)

setFont()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

setHelpMenu()-Methode

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

setLabel()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

setLayout()-Methode

-
- [Benutzerschnittstellen entwerfen](#)
- [Benutzerschnittstellen mit Swing](#)

setLookAndFeel()-Methode

-
- [Benutzerschnittstellen mit Swing](#)

setMenuBar()-Methode

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

setMnemonic()-Methode

-
- [Benutzerschnittstellen mit Swing](#)

setState()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

setText()-Methode

- [Einfache Benutzeroberflächen für Applets](#)
- [Einfache Benutzeroberflächen für Applets](#)
- [Einfache Benutzeroberflächen für Applets](#)
-
- [Ereignisbehandlung mit Swing](#)

setToolTipText()-Methode

-
- [Benutzerschnittstellen mit Swing](#)

setValue()-Methode

- [Einfache Benutzeroberflächen für Applets](#)

setVisible()-Methode

- [Benutzerschnittstellen mit Swing](#)

shiftDown()-Methode

- [Ereignisverarbeitung in Applets](#)

showConfirmDialog()-Methode

- [Benutzerschnittstellen mit Swing](#)

showDocument()-Methode

- [Kommunikation über das Internet](#)
- [JavaBeans und andere fortgeschrittene Features](#)

showInputDialog()-Methode

- [Benutzerschnittstellen mit Swing](#)

showMessageDialog()-Methode

- [Benutzerschnittstellen mit Swing](#)

showOptionDialog()-Methode

- [Benutzerschnittstellen mit Swing](#)

showStatus()-Methode

- [JavaBeans und andere fortgeschrittene Features](#)

Sicherheitsrichtlinien

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Signatures 'R' US

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

size()-Methode

- [Pakete, Schnittstellen und mehr](#)

Skeletons

- [JavaBeans und andere fortgeschrittene Features](#)

skip()-Methode

- [Java und Streams](#)

skipBytes()-Methode

-

[Java und Streams](#)

SocketImpl-Klasse

-

[Kommunikation über das Internet](#)

Socket-Klasse

-

[Kommunikation über das Internet](#)

Sockets

Erstellen



[Kommunikation über das Internet](#)

Ports



[Kommunikation über das Internet](#)

Serverseitig



[Kommunikation über das Internet](#)

Softwarekomponenten

- [JavaBeans und andere fortgeschrittene Features](#)

Solaris SPARC

-

[Eine Handvoll Java](#)

Solaris x86

-

[Eine Handvoll Java](#)

Speichermanagement

-

[Arbeiten mit Objekten](#)

Sphere-Klasse

-

[Pakete, Schnittstellen und mehr](#)

Spots-Applet

-

[Ereignisverarbeitung in Applets](#)

SQL, JDBC

- [JavaBeans und andere fortgeschrittene Features](#)
- [JavaBeans und andere fortgeschrittene Features](#)

sRGB

- [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Standarddialogfelder

- [Benutzerschnittstellen mit Swing](#)
 - ConfirmDialog
 - [Benutzerschnittstellen mit Swing](#)
 - InputDialog
 - [Benutzerschnittstellen mit Swing](#)
 - MessageDialog
 - [Benutzerschnittstellen mit Swing](#)
 - OptionDialog
 - [Benutzerschnittstellen mit Swing](#)

Standardtasten, Event-Klasse

- [Ereignisverarbeitung in Applets](#)

Star Trek

- [Eine Handvoll Java](#)

Star7

- [Eine Handvoll Java](#)
- [Kommunikation über das Internet](#)

Statement

- [Das Java-ABC](#)

Statuszeilen, Browser, Meldungen anzeigen

- [JavaBeans und andere fortgeschrittene Features](#)

stop()-Methode

- [Bilder, Sound und Animation](#)

Streamfilter verschachteln

-

[Java und Streams](#)

StreamTokenizer-Klasse

-

[Java und Streams](#)

String-Arithmetik

-

[Das Java-ABC](#)

StringSelection-Klasse

- [JavaBeans und andere fortgeschrittene Features](#)

Structured Query Language

- [JavaBeans und andere fortgeschrittene Features](#)

Stub-Implementierungen

-

[Pakete, Schnittstellen und mehr](#)

Stubs

- [JavaBeans und andere fortgeschrittene Features](#)

Subclassing

- [Objektorientierte Programmierung - ein erster Eindruck](#)

Subklasse

- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Objektorientierte Programmierung - ein erster Eindruck](#)

-

[Java: Eine Klassensprache](#)

-

[Pakete, Schnittstellen und mehr](#)

erstellen

- [Objektorientierte Programmierung - ein erster Eindruck](#)

SunSoft Java WorkShop

-

[Eine Handvoll Java](#)

SuperCede

-

[Eine Handvoll Java](#)

Superklasse

- [Objektorientierte Programmierung - ein erster Eindruck](#)
- [Objektorientierte Programmierung - ein erster Eindruck](#)

-

[Pakete, Schnittstellen und mehr](#)

Swing

- [Eine Handvoll Java](#)

SwingColorControls-Klasse

- [Benutzerschnittstellen mit Swing](#)
- [Ereignisbehandlung mit Swing](#)

SwingColorTest-Klasse

- [Benutzerschnittstellen mit Swing](#)
- [Ereignisbehandlung mit Swing](#)

SwingConstants-Klasse

- [Benutzerschnittstellen mit Swing](#)

Symantec Visual Caf

- [Eine Handvoll Java](#)

T

target-Instanzvariable

- [Ereignisverarbeitung in Applets](#)

Tastaturereignisse

- [Ereignisverarbeitung in Applets](#)
- [Ereignisbehandlung mit Swing](#)

Tastaturkrzel

- [Benutzerschnittstellen mit Swing](#)
- [Benutzerschnittstellen mit Swing](#)

Tastaturshortcut

- [Benutzerschnittstellen mit Swing](#)

TextArea-Klasse

- [Einfache Benutzeroberflächen für Applets](#)

TextComponent-Klasse

- [Einfache Benutzeroberflächen für Applets](#)

TextDialog-Klasse

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

Texteditor fr JDK

- [Texteditoren und das JDK](#)
 - DOS Edit
 - [Texteditoren und das JDK](#)
 - Windows Editor
 - [Texteditoren und das JDK](#)
 - Windows WordPad
 - [Texteditoren und das JDK](#)

TextField-Klasse

- [Einfache Benutzeroberflächen für Applets](#)

Textverarbeitungsprogramm

- [Texteditoren und das JDK](#)

Thawte Certification

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

theList()-Methode

- [Pakete, Schnittstellen und mehr](#)

Thread

- [Bilder, Sound und Animation](#)

Throwable-Klasse

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

throws-Anweisung

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

toCharArray()-Methode

- [Java und Streams](#)

Tool, jar

- [Grundlagen der Java-Applets](#)

ToolTips

- [Benutzerschnittstellen mit Swing](#)
- [Benutzerschnittstellen mit Swing](#)

Top-Level-Klassen

- [Pakete, Schnittstellen und mehr](#)

toString()-Methode

- [Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)
- [Ereignisbehandlung mit Swing](#)

TriverServer-Klasse

- [Kommunikation über das Internet](#)
- [Kommunikation über das Internet](#)

Trivia-Anwendung

- [Kommunikation über das Internet](#)
- [Kommunikation über das Internet](#)

Client

- [Kommunikation über das Internet](#)

U

UDP-Sockets

- [Kommunikation über das Internet](#)

UIManager-Klasse

- [Benutzerschnittstellen mit Swing](#)

Unicode

- [Das Java-ABC](#)
- Escapecodes
 - [Das Java-ABC](#)

Uniform Resource Locator

- [Bilder, Sound und Animation](#)

unread()-Methode

- [Java und Streams](#)

update()-Methode

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

URL

- [Bilder, Sound und Animation](#)

URL-Klasse

- [Kommunikation über das Internet](#)
- [JavaBeans und andere fortgeschrittene Features](#)

V

Variable

- [Das Java-ABC](#)
 - erstellen
 - [Das Java-ABC](#)
 - final
 - [Pakete, Schnittstellen und mehr](#)
 - Gltigkeitsbereich
 - [Java: Eine Klassesprache](#)
 - lokale
 - [Pakete, Schnittstellen und mehr](#)
 - Reflection
 - [Arbeiten mit Objekten](#)
 - [Arbeiten mit Objekten](#)
 - Zugriffskontrolle



[Pakete, Schnittstellen und mehr](#)

Vektor, Definition



[Java und Streams](#)

Vending



[Pakete, Schnittstellen und mehr](#)

Vererbung



[Objektorientierte Programmierung - ein erster Eindruck](#)



[Objektorientierte Programmierung - ein erster Eindruck](#)

Vererbungs-Hierarchie



[Pakete, Schnittstellen und mehr](#)

Verhaltensweisen



[Objektorientierte Programmierung - ein erster Eindruck](#)

VeriSign



[Ausnahmezustände: Fehlerbehandlung und Sicherheit](#)

Verknüpfungen, Applets erstellen



[JavaBeans und andere fortgeschrittene Features](#)

Verlaufsfillung



[Programme mit Grafik, Fonts und Farbe verfeinern](#)

azyklischer Verlauf

■ [Programme mit Grafik, Fonts und Farbe verfeinern](#)

zyklischer Verlauf

■ [Programme mit Grafik, Fonts und Farbe verfeinern](#)

Vernetzung



[Kommunikation über das Internet](#)



[Kommunikation über das Internet](#)

Visual Basic



[Eine Handvoll Java](#)

Visual C++



[Eine Handvoll Java](#)

WebRunner



[Eine Handvoll Java](#)

Webseiten

Applets, mehrere

- [JavaBeans und andere fortgeschrittene Features](#)
- [JavaBeans und andere fortgeschrittene Features](#)

JavaSoft

- [JavaBeans und andere fortgeschrittene Features](#)

Web-Standorte

Datenbertragungsspezifikation

- [JavaBeans und andere fortgeschrittene Features](#)

GetRaven-Applet



[Kommunikation über das Internet](#)

JDBC

- [JavaBeans und andere fortgeschrittene Features](#)

MIME-Typen

- [JavaBeans und andere fortgeschrittene Features](#)

while-Schleife



[Arrays, Bedingungen und Schleifen](#)

Wiederverwendung

- [Objektorientierte Programmierung - ein erster Eindruck](#)

WINDOW DEICONIFY Ereignis

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

WINDOW DESTROY Ereignis

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

WINDOW EXPOSE Ereignis

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

WINDOW ICONIFY Ereignis

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

WINDOW MOVED Ereignis

- [Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

windowActivated()-Methode



[Ereignisbehandlung mit Swing](#)

windowClosed()-Methode



[Ereignisbehandlung mit Swing](#)

windowClosing()-Methode



[Ereignisbehandlung mit Swing](#)

windowDeactivated()-Methode



[Ereignisbehandlung mit Swing](#)

windowDeiconified()-Methode



[Ereignisbehandlung mit Swing](#)

windowIconified()-Methode



[Ereignisbehandlung mit Swing](#)

Window-Klasse



[Fortgeschrittene Benutzeroberflächen mit dem AWT](#)

WindowListener-Schnittstelle



[Ereignisbehandlung mit Swing](#)

windowOpened()-Methode



[Ereignisbehandlung mit Swing](#)

Windows_95



[Eine Handvoll Java](#)

Windows_NT



[Eine Handvoll Java](#)

write()-Methode



[Java und Streams](#)



[Java und Streams](#)

Writer-Klasse



[Java und Streams](#)

writeTo()-Methode



Java und Streams

WWW, Verbindungen erstellen

- Kommunikation über das Internet
- Kommunikation über das Internet

X

x- und y-Argumente in Ereignissen

- Ereignisverarbeitung in Applets

Z

Zeichnen

Bgen

- Programme mit Grafik, Fonts und Farbe verfeinern

Linien

- Programme mit Grafik, Fonts und Farbe verfeinern

Ovale

- Programme mit Grafik, Fonts und Farbe verfeinern

Polygone

- Programme mit Grafik, Fonts und Farbe verfeinern

Rechtecke

- Programme mit Grafik, Fonts und Farbe verfeinern

Zeiger

- Arbeiten mit Objekten

Zertifikat

- Ausnahmezustände: Fehlerbehandlung und Sicherheit

Zertifizierungsautoritt

- Ausnahmezustände: Fehlerbehandlung und Sicherheit

Zugriffskontrollliste

- Grundlagen der Java-Applets

Ein Imprint des Markt+Technik Buch- und Software-Verlag GmbH.
Elektronische Fassung des Titels: Java 2 in 21 Tagen, ISBN: 3-8272-6014-0