

Der Inhalt (jetzt ausführlich)

E

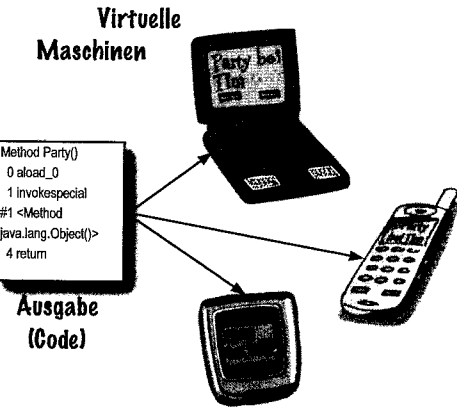
Einführung

Ihr Gehirn und Java. Sie versuchen, etwas zu *lernen*, und Ihr *Hirn* tut sein Bestes, damit das Gelernte nicht *hängen bleibt*. Es denkt nämlich: »Wir sollten lieber ordentlich Platz für wichtigere Dinge lassen, z.B. für das Wissen, welche Tiere einem gefährlich werden könnten, oder dass es eine ganz schlechte Idee ist, nackt Snowboard zu fahren.« Tja, *wie* schaffen wir es nun, Ihr Gehirn davon zu überzeugen, dass Ihr Leben davon abhängt, etwas über Java zu wissen?

Für wen ist dieses Buch?	xviii
Wir wissen, was Ihr Gehirn denkt	xix
Metakognition	xxi
Machen Sie sich Ihr Hirn untertan	xxiii
Was Sie für dieses Buch brauchen	xxiv
Die Fachgutachter	xxvi
Danksagungen	xxvii

1 Die Oberfläche durchbrechen

Java führt Sie an neue Orte. Seit seinem bescheidenen Schritt an die Öffentlichkeit mit der (kümmerlichen) Version 1.02 hat Java mit seiner freundlichen Syntax, seinen objektorientierten Features, der Speicherverwaltung und natürlich mit dem Versprechen der Portabilität Programmierer verführt. Wir machen mal eine kleine Kostprobe und schreiben ein bisschen Code, kompilieren ihn und lassen ihn laufen. Wir sprechen über die Syntax, Schleifen, Verzweigungen und alles, was Java so cool macht. Springen Sie rein!

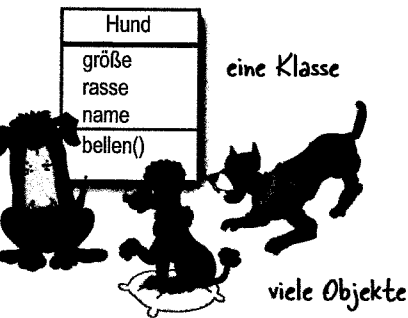


Wie Java funktioniert	2
Code-Struktur in Java	7
Anatomie einer Klasse	8
Die main-Methode	9
Schleifen (<i>if</i> -Tests)	11
Bedingte Verzweigungen	13
»99 Bierflaschen« programmieren	14
Phras-O-Mat	16
Kamingespräche: Der Compiler und die JVM	18
Übungen und Rätsel	20

2 Ein Ausflug nach Objekthausen

Man hat mir gesagt, wir hätten es hier mit Objekten zu tun.

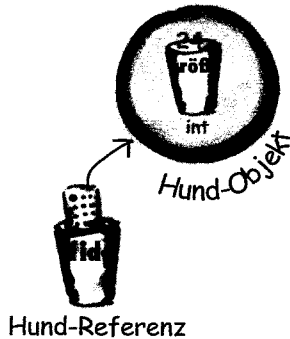
Aber in Kapitel 1 haben wir unseren gesamten Code in die `main()`-Methode gesteckt. Das ist nicht sonderlich objektorientiert! Jetzt müssen wir diese prozedurale Welt hinter uns lassen und damit beginnen, eigene Objekte zu machen. Wir werden uns ansehen, was die objektorientierte Entwicklung in Java zu einem solchen Vergnügen macht. Wir werden uns den Unterschied zwischen einer Klasse und einem Objekt ansehen. Wir werden uns ansehen, wie Objekte Ihr Leben verbessern.



Stuhlkrige (Stefan, der OO-Typ, gegen Harry, den prozeduralen Programmierer)	28
Vererbung (eine Einführung)	31
Methoden überschreiben (eine Einführung)	32
Was ist in einer Klasse? (Methoden, Instanzvariablen)	34
Ein erstes Objekt machen	36
<code>main()</code> verwenden	38
Der Code für das Ratespiel	39
Übungen und Rätsel	42

3 Verstehen Sie Ihre Variablen

Variablen gibt es in zwei Geschmacksrichtungen: elementare Typen und Referenztypen. Im Leben muss es doch mehr geben als Integer, Strings und Arrays. Was ist, wenn Sie ein HaustierBesitzer-Objekt mit einer Hund-Instanzvariablen haben? Oder ein Auto mit einem Motor? In diesem Kapitel werden wir die Mysterien von Java-Typen offen legen und uns ansehen, was Sie als Variable deklarieren können, was Sie in eine Variable stecken können und was Sie mit einer Variablen tun können. Und schließlich werden wir verstehen, wie das Leben, das sich auf dem Garbage Collectible Heap abspielt, wirklich ist.



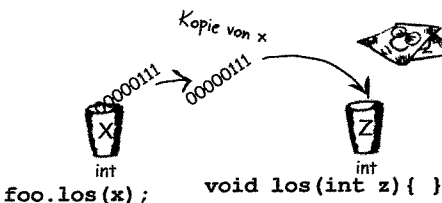
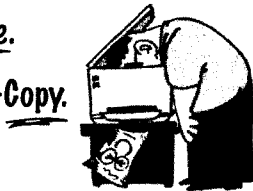
Eine Variable deklarieren (Java kümmert sich um Typen)	50
Elementare Typen (»Einen doppelten Espresso, bitte!«)	51
Java-Schlüsselwörter	53
Referenzvariablen (Fernsteuerung für ein Objekt)	54
Objekte deklarieren und zuweisen	55
Objekte auf dem Garbage Collectible Heap	57
Arrays (ein erster Blick)	59
Übungen und Rätsel	63

4 Wie sich Objekte verhalten

Zustände haben Auswirkungen auf Verhalten, und Verhalten hat Auswirkungen auf Zustände. Wir wissen, dass Objekte Zustände und Verhalten haben, die von Instanzvariablen und Methoden repräsentiert werden. Nun sehen wir uns an, in welcher Beziehung Zustände und Verhalten zueinander stehen. Objekte haben ein Verhalten, das auf ihre Zustände wirkt. Anders gesagt, Methoden nutzen die Werte von Instanzvariablen – beispielsweise »wenn Hund weniger wiegt als 7 Kilo, mach Kläffgeräusch« oder »erhöhe Gewicht um 3 Kilo«. Ändern wir mal ein paar Zustände.

Java ist Pass-by-Value.

Das bedeutet Pass-by-Copy.

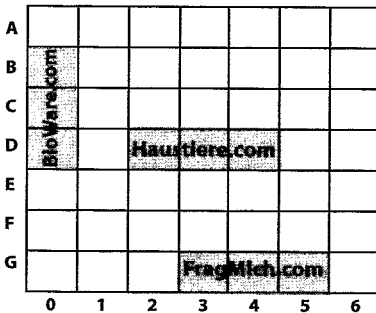


Methoden nutzen Objektzustände (Die Größe beeinflusst das Bellen)	73
Methoden-Argumente und Rückgabetypen	74
Pass-by-Value (Die Variable wird immer kopiert)	77
Getter und Setter	79
Kapselung (tun Sie es oder machen Sie sich lächerlich)	80
Referenzen in Arrays verwenden	83
Übungen und Rätsel	88

5 Methoden mit Superkraft

Jetzt werden wir unseren Methoden etwas mehr Muskelkraft geben. Wir haben uns an Variablen versucht, haben mit ein paar Objekten gespielt und haben etwas Code geschrieben. Jetzt brauchen wir mehr Werkzeuge. Wie **Operatoren**. Und **Schleifen**. Es könnte auch nützlich sein, **Zufallszahlen zu generieren**. Oder **einen String in ein int umzuwandeln**, ja, das wäre cool. Und warum lernen wir nicht all den Kram, während wir was Echtes *schreiben*, um zu erfahren, wie es ist, wenn man ein Programm von der Pike auf schreiben (und testen) muss? **Vielleicht ein Spiel**, in dem wir Dot-Coms abschießen (wie bei Schiffe versenken).

Wir werden das Spiel
»Dot-Coms versenken«
entwickeln



»Dot-Com versenken« entwickeln	96
EinfachesDotComSpiel (die simple Version)	98
Vorcode für EinfachesDotComSpiel schreiben	100
Testcode für EinfachesDotComSpiel schreiben	102
EinfachesDotComSpiel schreiben	103
Endgültigen Code für EinfachesDotComSpiel schreiben	106
Zufallszahlen mit Math.random() generieren	111
Code-Fertiggericht für die Benutzereingabe in der Kommandozeile	112
for-Schleifen	114
Elementare Typen von großer Größe zu kleiner Größe casten	117
Einen String mit Integer.parseInt() in ein int umwandeln	117
Übungen und Rätsel	118

6 Die Java-Bibliothek verwenden

Java wird mit Hunderten vorgefertigter Klassen ausgeliefert.

Sie müssen das Rad nicht jedes Mal neu erfinden, wenn Sie wissen, wie Sie das, was Sie brauchen, in der Java-Bibliothek finden, die als die **Java-API** bezeichnet wird. *Schließlich haben Sie etwas Besseres zu tun*. Wenn Sie Code schreiben, können Sie sich ruhig auf die Teile beschränken, die bei Ihrer Anwendung besonders sind. Die Java-Kernbibliothek ist ein gigantischer Haufen von Klassen, die nur darauf warten, dass Sie sie als Bausteine einsetzen.

»Gut zu wissen, dass es im Package
java.util eine ArrayList gibt. Aber wie
hätte ich das allein rausfinden können?«

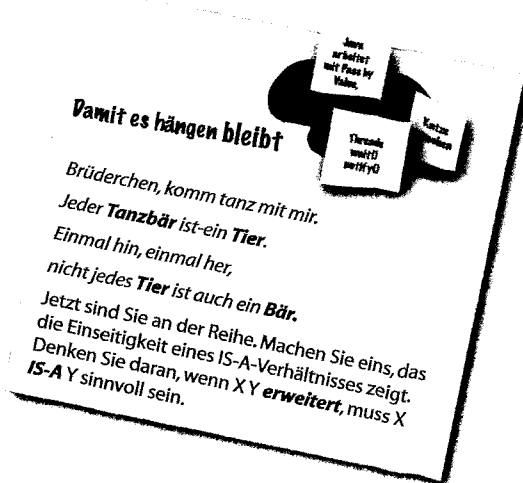
- Julia, 31, Hand-Model



Den Fehler in unserem Dot-Com-Spiel analysieren	126
ArrayList (sich die Vorteile der Java API zunutze machen)	132
Den DotCom-Code reparieren	138
Das richtige Spiel bauen (»Dot-Com versenken«)	140
Vorcode für das richtige Spiel	144
Code für das richtige Spiel	146
Boolesche Ausdrücke	151
Die Bibliothek (die Java-API) verwenden	154
Packages verwenden (Import-Anweisungen und vollständige Namen)	155
Die HTML-API-Dokumentation und Referenzbücher nutzen	158
Übungen und Rätsel	161

7 Besser leben in Objekthausen

Planen Sie Ihre Programme mit Blick auf die Zukunft Was wäre, wenn Sie Code schreiben könnten, den ein *anderer* **problemlos** erweitern kann? Und was wäre, wenn Ihr Code flexibel genug für diese verfluchten Spezifikationsänderungen in letzter Minute wäre? Wenn Sie auf den Polymorphie-Zug springen, lernen Sie die fünf Schritte zu besserem Klassen-Design, die drei Tricks der Polymorphie und die acht Wege, Code flexibel zu machen. Dazu, wenn Sie sofort zugreifen – eine Bonusstunde mit den vier Tipps zur Nutzung von Vererbung inklusive

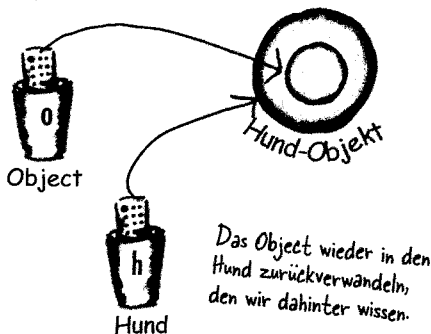


Vererbung verstehen (Unterklassen und Superklassen)	168
Einen Vererbungsbaum entwerfen (eine Tier-Simulation)	170
Codeverdopplung verhindern (mit Vererbung)	171
Methoden überschreiben	170
IST-EIN und HAT-EIN (die Badewannen-Frau)	177
Was kann man von der Superklasse erben?	180
Was <i>bringt</i> der ganze Vererbungsbaum?	182
Polymorphie (eine Supertyp-Referenz für Unterklassen-Objekt verwenden)	183
Regeln fürs Überschreiben (lassen Sie die Finger von den Argumenten und Rückgabetypen!)	190
Eine Methode überladen (nichts weiter als Methodennamen wiederverwenden)	191
Übungen und Rätsel	192

8 Ernsthafte Polymorphie

Vererbung ist nur der Anfang. Um Polymorphie richtig zu nutzen, brauchen wir Schnittstellen. Wir müssen über einfache Vererbung hinausgehen und eine Stufe der Flexibilität erreichen, die man nur erhält, wenn man Schnittstellendefinitionen als Ausgangsbasis für den Entwurf und die Programmierung nimmt. Was eine Java-Schnittstelle ist? Eine 100% abstrakte Klasse. Was eine abstrakte Klasse ist? Das ist eine Klasse, die nicht instantiiert werden kann. Und wozu soll die gut sein? Lesen Sie das Kapitel...

```
Object o = al.get(index);  
Hund h = (Hund) o;  
h.bewegen();
```

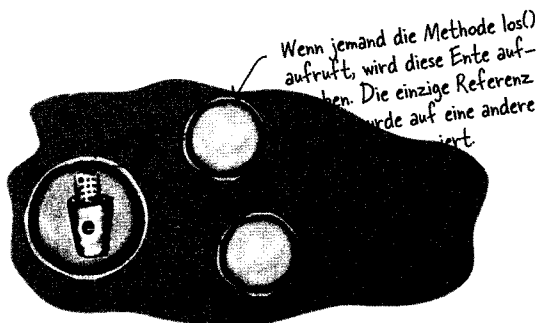


Manche Klassen sollten einfach <i>nicht</i> instantiiert werden	200
Abstrakte Klassen (<i>können nicht</i> instantiiert werden)	201
Abstrakte Methoden (müssen implementiert werden)	203
Polymorphie in Aktion	206
Die Klasse Object (die ultimative Superklasse für <i>alles</i>)	208
Objekte aus einer ArrayList nehmen (sie kommen als Typ Object raus)	211
Der Compiler prüft den Referenztyp (bevor er Sie eine Methode aufrufen lässt)	213
Finden Sie Ihr inneres Object	214
Polymorphe Referenzen	215
Eine Objektreferenz casten (im Vererbungsbaum nach unten verschieben)	216
»The Deadly Diamond of Death« (Problem der Mehrfachvererbung)	223
Interfaces verwenden (die beste Lösung)	224
Übungen und Rätsel	230



9 Konstruktoren und Garbage Collection

Objekte werden geboren und Objekte sterben. Sie herrschen über das Leben eines Objekts. Sie entscheiden, wie und wann ein Objekt *konstruiert* wird. Sie entscheiden, wann es aufgegeben werden soll. Der **Garbage Collector (gc)** will den Speicher wieder haben. In diesem Kapitel sehen wir uns an, wie Objekte erzeugt werden, wie sie leben, während sie leben, und wie Sie sie effektiv bewahren oder aufgeben. Das bedeutet, dass wir über den Heap, den Stack, Geltungsbereiche, Konstruktoren, Super-Konstruktoren, null-Referenzen und anderes reden werden.



>>> erhält ein neues Ente-Objekt zugewiesen und gibt damit das ursprüngliche (erste) Ente-Objekt auf. Die erste Ente ist jetzt so gut wie tot.

Der Stack und der Heap: wo das Leben spielt	236
Methoden auf dem Stack	237
Wo die <i>lokalen</i> Variablen leben	238
Wo die <i>Instanzvariablen</i> leben	239
Das Wunder der Objekterzeugung	240
Konstruktoren (der Code, der ausgeführt wird, wenn Sie <i>new</i> sagen)	241
Den Zustand einer neuen Ente initialisieren	243
Überladene Konstruktoren	247
Superklassenkonstruktoren (Konstruktorverkettung)	250
Einen überladenen Konstruktor mit <i>this()</i> aufrufen	256
Das Leben eines Objekts	258
Garbage Collection (und Objekte auswählbar machen)	260
Übungen und Rätsel	266

10 Zahlen, bitte!

Rechnen Sie's aus. In der Java-API gibt es eine Menge praktischer Methoden für Absolutbeträge, zum Runden, zur Extremwertbestimmung etc. Aber wie sieht es mit der Formatierung aus? Vielleicht möchten Sie, dass Ihre Zahlen mit exakt zwei Nachkommastellen ausgegeben oder dass große Zahlen mit Trennzeichen unterteilt werden, damit man sie leichter lesen kann. Und wie steht es mit Datumsangaben? Als Erstes sehen wir uns einmal an, was es für eine Variable oder eine Methode bedeutet, statisch zu.

Alle Instanzen der gleichen Klasse teilen sich ein einziges Exemplar der statischen Variablen.



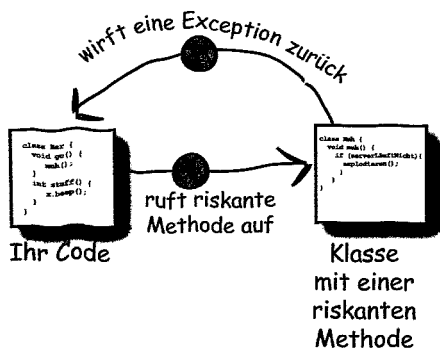
Instanzvariablen:
eine pro Instanz

statische Variablen:
eine pro Klasse

Die Klasse Math (brauchen wir davon eine Instanz?)	274
Statische Methoden	275
Statische Variablen	277
Konstanten (statische finale Variablen)	282
Math-Methoden (<code>random()</code> , <code>round()</code> , <code>abs()</code> usw.)	286
Wrapperklassen (Integer, Boolean, Character usw.)	287
Autoboxing	289
Zahlen formatieren	294
Formatierung und Manipulation von Datumsangaben	301
Statische Importe	307
Übungen und Rätsel	310

11 Riskantes Verhalten

Manches passiert einfach. Die Datei ist nicht da. Der Server läuft nicht. Und wenn Sie ein noch so guter Programmierer sind – Sie können nicht *alles* unter Kontrolle haben. Wenn Sie eine riskante Methode schreiben, brauchen Sie Code, der all das Schlimme, das möglicherweise passiert, behandelt. Aber woher *wissen* Sie, wann eine Methode riskant ist? Und wo platzieren Sie den Code, der die **Ausnahmesituation** *behandelt*? In *diesem* Kapitel programmieren wir einen MIDI-Musikplayer, der die riskante Java-Sound-API benutzt – daher sollten wir das besser alles schnell rausfinden!



Bauen wir uns eine Musikmaschine	316
Wenn eine Methode, die Sie aufrufen wollen, riskant ist ...	319
Exceptions sagen: »Es ist etwas Schlimmes passiert.«	320
Der Compiler garantiert (er <i>prüft</i>), dass Sie das Risiko kennen	321
Das <i>try-and-catch</i> -Prinzip (Skateboarder)	322
Flusskontrolle in <i>try/catch</i> -Blöcken	326
Der <i>finally</i> -Block (»egal, was passiert, schalte den Herd aus!«)	327
Abfangen von Mehrfach-Exceptions	329
Eine Exception deklarieren (weichen Sie ihr einfach aus!)	335
Das Gesetz »Behandeln oder deklarieren«	337
CodeKüche (Sound erzeugen)	339
Übungen und Rätsel	348

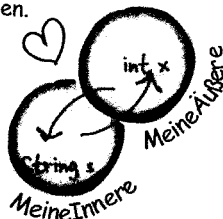
12 Innen hui, außen GUI

Sehen Sie den Tatsachen ins Auge: Um GUIs (grafische Benutzeroberflächen) kommen Sie nicht herum. Selbst wenn Sie davon ausgehen, dass Sie den Rest Ihres Lebens serverseitigen Code schreiben, müssen Sie früher oder später Tools entwickeln, und dafür hätten Sie dann ganz sicher gern eine grafische Schnittstelle. Wir werden zwei Kapitel mit der Programmierung von GUIs zubringen und dabei grundlegende Features der Sprache Java kennen lernen, z.B. **Event-Handling** und **innere Klassen**. In diesem Kapitel setzen wir einen Button auf den Bildschirm und lassen ihn reagieren, wenn man ihn anklickt. Außerdem zeichnen wir etwas auf den Bildschirm, zeigen ein JPEG-Bild an und versuchen uns sogar an einer kleinen Animation.

```
class MeineÄußereKlasse {
    class MeineInnereKlasse {
        void los() {
        }
    }
}
```

Innere Klasse ist vollständig von äußerer Klasse umschlossen.

Diese beiden Objekte auf dem Heap haben eine ganz besondere Bindung aneinander. Das innere kann die Variablen des äußeren benutzen (und umgekehrt).



Ihre erste GUI	355
Wie man an ein Benutzerereignis kommt	357
Ein Listener-Interface implementieren	358
Wie man an das ActionEvent eines Buttons kommt	360
Dinge in eine GUI setzen	363
Schönes mit <i>paintComponent()</i>	365
Das <i>Graphics2D</i> -Objekt	366
Zwei oder mehr Widgets in einen Frame setzen	370
Innere Klassen kommen uns zu Hilfe	376
Animation	382
CodeKüche (Grafiken zum Beat der Musik einfügen)	386
Übungen und Rätsel	394

13 Mehr Schwung mit Swing

Swing ist einfach. Wenn es Ihnen nicht so wichtig ist, wo die Dinge auf dem Bildschirm landen. Swing-Code *sieht einfach aus*, aber dann kompilieren Sie das Ganze, führen es aus, sehen es sich an und denken: »Hey, das sollte aber nicht *dort* stehen ...« Der Grund dafür, dass der Code so leicht zu *schreiben* ist, ist der **Layoutmanager** – aber genau der macht es auch so *schwer kontrollierbar*. Mit ein bisschen Mühe bringen Sie Layoutmanager jedoch dazu, sich Ihrem Willen zu unterwerfen. In diesem Kapitel bringen wir unsere Swing-Künste in Schwung und lernen etwas über Widgets.

Komponenten im Osten und Westen bekommen ihre bevorzugte Breite.

Komponenten im Norden und Süden bekommen ihre bevorzugte Höhe.



Swing-Komponenten	400
Layoutmanager steuern Größe und Platzierung	401
Drei Layoutmanager: Border, Flow und Box	403
BorderLayout (managt fünf Bereiche)	404
FlowLayout (managt bevorzugte Größe und die Reihenfolge)	408
BoxLayout (ähnelt FlowLayout, kann Komponenten vertikal anordnen)	411
JTextField (für einzeilige Benutzereingaben)	413
JTextArea (für mehrzeilige Benutzereingaben, Scrollen)	414
JCheckBox (angeklickt?)	416
JList (scrollbare, auswählbare Liste)	417
CodeKüche (Ein Ungetüm – den BeatBox-Chat-Clients erzeugen)	418
Übungen und Rätsel	424

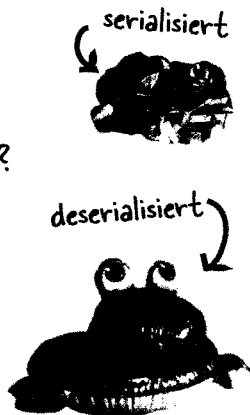
14 Speicherung von Objekten

Objekte lassen sich flach drücken und wieder aufblasen.

Objekte haben einen Zustand und ein Verhalten. Das *Verhalten* steckt in der *Klasse*, aber der *Zustand* steckt in jedem einzelnen *Objekt*. Muss Ihr Programm Zustände speichern, können Sie das natürlich auf die mühsame Weise tun – Sie befragen jedes Objekt und notieren dann akribisch den Wert jeder Instanzvariablen. Oder **Sie machen es auf die einfache, objektorientierte Weise** – indem Sie lediglich das Objekt selbst gefriertrocknen (serialisieren) und es mit Wasser anrühren (deserialisieren), wenn Sie es zurückhaben wollen.

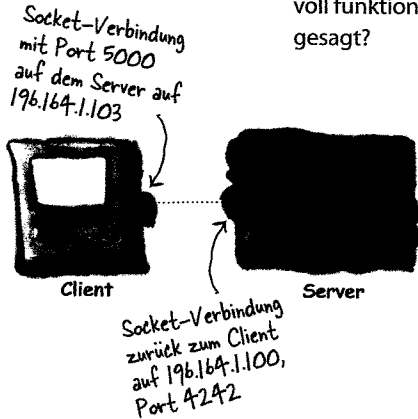
Speicherung von Zuständen	431
Ein serialisiertes Objekt in eine Datei schreiben	432
Input- und Outputströme (Anschluss und Verkettung)	433
Serialisierte Objekte	434
Das Serializable-Interface implementieren	437
Transiente Variablen verwenden	439
Ein Objekt deserialisieren	441
In eine Textdatei schreiben	447
Die Klasse java.io.File	452
Aus einer Textdatei lesen	454
Einen String mit split() in mehrere Tokens spalten	458
CodeKüche	462
Übungen und Rätsel	466

Noch Fragen?



15 Eine Verbindung herstellen

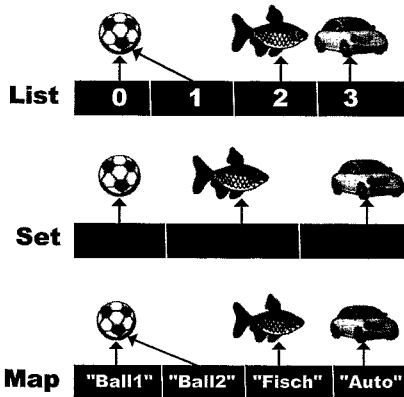
Nehmen Sie Verbindung mit der Außenwelt auf. Das ist nicht schwer. Um all die Lowlevel-Netzwerk-Details kümmern sich Klassen in der `java.net`-Bibliothek. Einer der großen Vorteile von Java ist, dass das Senden und Empfangen über ein Netz einfache Eingabe/Ausgabe ist, lediglich mit einem etwas anderen Verbindungsstrom am Ende der Kette. In diesem Kapitel machen wir *Client-Sockets*. Wir machen *Server-Sockets*. Wir machen *Clients* und *Server*. Und wir lassen sie miteinander reden. Bevor Sie mit dem Kapitel durch sind, haben Sie einen voll funktionsfähigen und Multithread-tauglichen Chat-Client. Haben wir da gerade *Multithread* gesagt?



Das Chat-Programm im Überblick	473
Verbinden, senden und empfangen	474
Netzwerk-Sockets	475
TCP-Ports	476
Daten von einem Socket lesen (mit <code>BufferedReader</code>)	478
Daten in einen Socket schreiben (mit <code>PrintWriter</code>)	479
Das <code>TippDesTagesClient</code> -Programm schreiben	480
Einen einfachen Server schreiben	483
Der <code>TippDesTagesServer</code> -Code	484
Einen Chat-Client schreiben	486
Mehrere Aufruf-Stacks	490
Einen neuen Thread starten	492
Das <code>Runnable</code> -Interface (Job des Threads)	494
Drei Zustände eines neuen Threads (neu, lauffähig, laufend)	495
Die lauffähig/laufend-Schleife	496
Der Thread-Scheduler (das ist seine Entscheidung, nicht Ihre!)	497
Einen Thread schlafen legen	501
Erzeugen und Starten von zwei Threads	503
Ehe in Gefahr: Ist dieses Paar noch zu retten?	505
Das Rainer-und-Monika-Problem in Codeform	506
Dinge sperren, um sie atomar zu machen	510
Jedes Objekt hat ein Schloss	511
Das gefürchtete »Problem des verlorenen Updates«	512
Synchronisierte Methoden (eine Sperre verwenden)	514
Thread-Deadlock	516
ChatClient mit mehreren Threads	518
Code-Fertigericht für den <code>EinfacherChatServer</code>	520
Übungen und Rätsel	524

16 Datenstrukturen

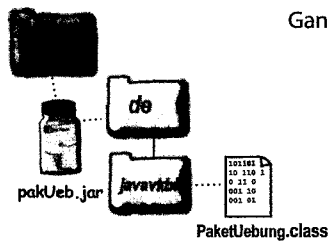
Sortieren ist in Java ein Klacks. Sie haben alle Werkzeuge zur Verfügung, um Daten zu sammeln und damit zu arbeiten, ohne Ihre eigenen Sortieralgorithmen schreiben zu müssen. Das Collections-Framework von Java enthält eine Datenstruktur für praktisch alles, was Sie jemals tun müssen. Sie wollen eine Liste, zu der Sie leicht etwas hinzufügen können? Sie wollen etwas mit Hilfe seines Namens suchen? Oder Sie brauchen eine Liste, die automatisch alles entfernt, was doppelt vorhanden ist? Sie möchten Ihre Kollegen danach sortieren, wie oft sie Ihnen in den Rücken gefallen sind? Es ist alles da ...



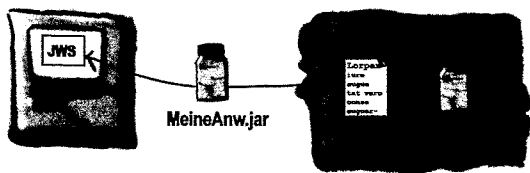
Collections	533
Eine ArrayList mit Collections.sort() sortieren	534
Generics und Typsicherheit	540
Sachen sortieren, die das Comparable-Interface implementieren	547
Mit einem eigenen Comparator sortieren	552
Die Collection-API (List, Set und Map)	557
Verdopplungen mit HashSet verhindern	559
Überschreiben von hashCode() und equals()	560
HashMap	567
Wildcardcards für Polymorphie verwenden	574
Übungen und Rätsel	576

17 Veröffentlichen Sie Ihren Code

Es ist Zeit loszulassen. Sie haben Ihren Code geschrieben. Sie haben ihn getestet. Sie haben ihn verbessert. Sie haben allen Bekannten erzählt, es wäre schön, wenn Sie nie wieder eine einzige Zeile Code sehen würden. Und doch haben Sie zu guter Letzt ein Kunstwerk geschaffen. Tatsächlich, das Ding läuft! Aber was nun? In den beiden letzten Kapiteln des Buchs befassen wir uns mit dem Organisieren, Packen und Deployment des Codes. Wir sehen uns lokale und entfernte Deployment-Möglichkeiten an, inklusive ausführbarer JARs, RMI und Servlets. Ganz ruhig – einige der coolsten Dinge, die Java bietet, sind einfacher, als Sie denken.

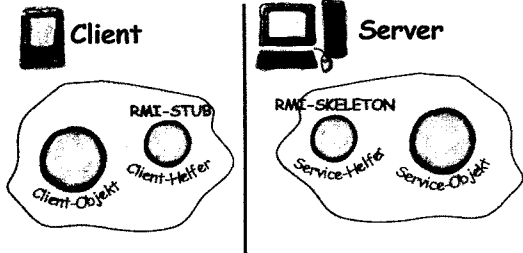


Deployment-Optionen	582
Quellcode und class-Dateien getrennt halten	584
Machen Sie das JAR ausführbar	585
Das JAR ausführen	586
Stecken Sie Ihre Klassen in Pakete!	587
Pakete müssen eine passende Verzeichnisstruktur haben	589
Kompilieren und Ausführen mit Paketen	590
Kompilieren mit -d	591
Ein ausführbares JAR mit Paketen erzeugen	592
Java Web Start (JWS) fürs Deployment aus dem Web	597
So erstellt man eine JWS-Anwendung	600
Übungen und Rätsel	601



18 Verteilte Anwendungen

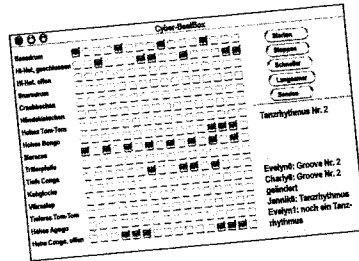
Entfernung muss kein Problem sein. Sicher, es ist einfacher, wenn alle Teile Ihrer Anwendung an einem Ort sind, auf einem Heap, mit einer JVM, die über alles herrscht. Aber das ist nicht immer möglich. Und auch nicht immer wünschenswert. Was ist, wenn Ihr Programm leistungsintensive Berechnungen durchführen soll? Was ist, wenn Ihr Programm Daten aus einer sicheren Datenbank braucht? In diesem Kapitel lernen wir, wie man die erstaunlich einfache Java-Technik namens Remote Method Invocation (RMI) einsetzt. Wir werfen auch einen kurzen Blick auf Servlets, Enterprise Java Beans (EJB) und Jini.



RMI zum Mitmachen, <i>sehr</i> ausführlich	614
Ein kurzer Blick auf Servlets	625
Ein <i>sehr</i> kurzer Blick auf Enterprise Java Beans (EJB)	631
Bezauberndes Jini	632
Den wirklich coolen Universaldienstbrowser erzeugen	636
Ende	648

A Anhang A

Das CodeKüchen-Finale! Der vollständige Code für unser BeatBox-Client-Programm – Ihre Chance, ein Rockstar zu werden!



Der Client-Code	650
Der Server-Code	657

B Anhang B

Die Top Ten der Themen, die es nicht ins Buch geschafft haben. Noch können wir Sie nicht ganz gehen lassen. Ein paar Sachen haben wir noch, aber dann sind Sie fertig. Diesmal meinen wir es ernst.

Die Top Ten	660
-------------	-----

I Index