Der Inhalt (im Überblick)

	Einführung	xix
1	Die Oberfläche durchbrechen: *Tauchen Sie ein: eine Kostprobe*	1
2	Die Reise nach Objectville: *Klassen und Objekte*	27
3	Verstehen Sie Ihre Variablen: *Elementare Datentypen und Referenzen*	49
4	Wie sich Objekte verhalten: *Methoden benutzen Instanzvariablen*	71
5	Methoden mit Superkräften: *Ein Programm schreiben*	95
6	Die Java-Bibliothek verwenden: *Die Java-API kennenlernen*	125
7	Besser leben in Objectville: *Vererbung und Polymorphie*	167
8	Ernsthafte Polymorphie: *Interfaces und abstrakte Klassen*	199
9	Leben und Sterben eines Objekts: *Konstruktoren und Garbage Collection*	237
10	Zahlen, bitte!: *Zahlen und Statisches*	*275*
11	Datenstrukturen: *Collections mit Generics*	309
12	Was – nicht wie!: *Lambdas und Streams*	369
13	Riskantes Verhalten: *Exception-Handling*	421
14	Innen hui, außen GUI: *Grafische Benutzeroberflächen*	461
15	Mehr Schwung mit Swing: *Swing im Einsatz*	509
16	Objekte (und Text) speichern: *Serialisierung und Datei-I/O*	539
17	Eine Verbindung herstellen: *Netzwerkprogrammierung und Threads*	587
18	Nebenläufigkeitsprobleme behandeln: *Konkurrenzprobleme und immutable Daten*	639
A	Anhang A: *Die finale Codeküche*	673
B	Anhang B: *Die (mehr als) zehn wichtigsten Themen, die es nicht ins Buch geschafft haben ...*	683
	Index	701

Der Inhalt (jetzt ausführlich)

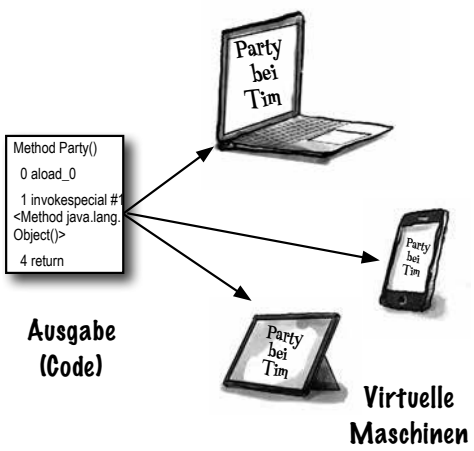
Einführung

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?	xx
Wir wissen, was Sie denken.	xxi
Und wir wissen, was Ihr *Gehirn* gerade denkt.	xxi
Das haben WIR getan	xxiv
Was Sie für dieses Buch brauchen	xxvi
Die Fachgutachter der dritten englischen Auflage	xxviii
Die Fachgutachter der zweiten englischen Auflage	xxx

1 Die Oberfläche durchbrechen

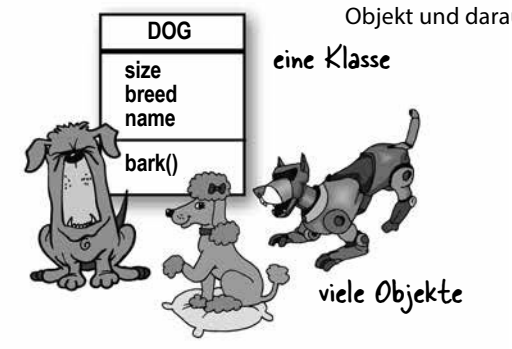
Java bringt Sie an neue Orte. Seit seiner bescheidenen Veröffentlichung in der (schwächlichen) Version 1.02 hat Java Programmierer mit seiner freundlichen Syntax, seinen objektorientierten Features, der Speicherverwaltung und vor allem mit dem Versprechen auf Portabilität verführt. Wir machen mal eine 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
Was Sie in Java tun werden	3
Eine sehr kurze Geschichte von Java	4
Codestruktur in Java	7
Eine Klasse mit einer main()-Methode schreiben	9
Einfache boolesche Tests	13
Bedingte Verzweigungen	15
Eine ernsthafte Geschäftsanwendung schreiben	16
Der Phras-O-Mat	19
Übungen	20

2 Die Reise nach Objectville

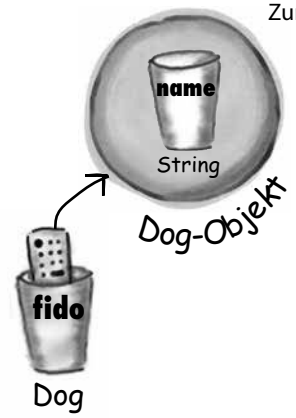
Ich dachte, hier gibt's Objekte. In Kapitel 1 haben wir den ganzen Code in die main()-Methode gepackt. Das ist jedoch nicht besonders objektorientiert – eigentlich überhaupt nicht. Aber jetzt ist endlich die Zeit gekommen, die prozedurale Welt hinter uns zu lassen. Nichts wie raus aus main() und ran an die Erstellung unserer eigenen Objekte! Wir werden erfahren, warum die objektorientierte Entwicklung in Java so viel Spaß macht. Außerdem werfen wir einen Blick auf den Unterschied zwischen einer Klasse und einem Objekt und darauf, wie Objekte Ihr Leben verbessern können.



Stuhlkriege	28
Ihr erstes Objekt erstellen	36
Die Film-Objekte erstellen und testen	37
Schnell! Nichts wie raus aus main()!	38
Das Ratespiel ausführen	40
Übungen	42

3 Verstehen Sie Ihre Variablen

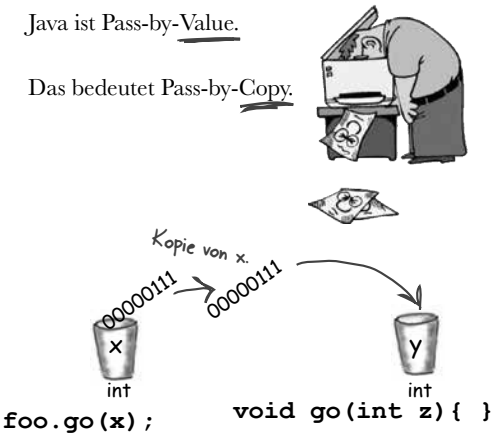
Variablen gibt es in zwei Geschmacksrichtungen: elementare Typen und Referenztypen. Das Leben kann doch nicht nur aus Integer-Werten, Strings und Arrays bestehen! Wie wäre es mit einem HaustierBesitzer-Objekt und einer Instanzvariablen vom Typ Hund? Oder mit einem Auto-Objekt, das einen Motor hat? In diesem Kapitel werden wir die Geheimnisse der Java-Typen lüften und uns ansehen, was man alles als Variablen *deklarieren* kann, was Sie in einer Variablen *speichern* und mit einer Variablen *tun* können. Zum Schluss lernen Sie das Leben auf dem Garbage Collectible Heap hautnah kennen.



Eine Variable deklarieren	50
»Einen doppelten Espresso bitte, ach nein, doch lieber einen int.«	51
Finger weg von Schlüsselwörtern!	53
Ein Dog-Objekt kontrollieren	54
Eine Objektreferenz ist einfach ein anderer Variablenwert.	55
Das Leben auf dem Garbage Collectible Heap	57
Ein Array ist wie ein Schrank voller Tassen … ähm Becher	59
Ein Dog-Beispiel	62
Übungen	63

4 Wie sich Objekte verhalten

Zustand beeinflusst Verhalten, Verhalten beeinflusst Zustände. Wir wissen, dass Objekte über **Zustand** und **Verhalten** verfügen, die durch **Instanzvariablen** und **Methoden** repräsentiert werden. Nun sehen wir uns an, in welcher Beziehung Zustände und Verhalten zueinander stehen. Objekte haben ein Verhalten, das sich auf ihre Zustände auswirkt. Anders gesagt, ***Methoden benutzen die Werte von Instanzvariablen.*** Ein Beispiel: »Wenn der Hund weniger als 7 Kilo wiegt, erzeuge ein Kläffgeräusch, sonst …« oder »erhöhe das Gewicht um 5«. ***Also los, lassen Sie uns ein paar Zustände ändern.***

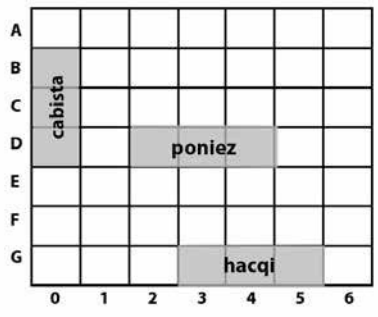


Erinnern Sie sich: Eine Klasse beschreibt, was ein Objekt weiß und was es tut.	72
Die Größe beeinflusst das Bellen	73
Sie können einer Methode Informationen schicken	74
Sie können von Methoden etwas zurückbekommen	75
Sie können einer Methode mehr als eine Sache übergeben	76
Parameter und Rückgabetypen	79
Kapselung	80
Wie verhalten sich Objekte in einem Array?	83
Instanzvariablen deklarieren und initialisieren	84
Variablen vergleichen (elementare und Referenztypen)	86
Übungen	88

5 Methoden mit Superkräften

Jetzt wollen wir unsere Methoden mal etwas aufmöbeln. Wir haben uns mit Variablen beschäftigt, mit ein paar Objekten herumgespielt und ein bisschen Code geschrieben. Aber wir brauchen Werkzeuge. Wie **Operatoren**. Und **Schleifen**. Sie könnten nützlich sein, um **Zufallszahlen zu erzeugen**. Oder **einen String in einen int umzuwandeln**, ja, das wäre cool. Und warum lernen wir das alles nicht, indem wir sofort etwas Reales bauen? Dann sehen wir, wie es ist, ein Programm von Grund auf neu zu coden (und zu testen). **Vielleicht ein Spiel** wie Schiffe versenken.

Lasst uns ein Startups-versenken-Spiel bauen!



Ein Spiel im Schiffe-versenken-Style: »Startups versenken«	96
Eine Klasse entwickeln	99
Die Methodenimplementierungen schreiben	101
Den Testcode für die SimpleStartup-Klasse schreiben	102
Die checkYourself()-Methode	104
Vorcode für die SimpleStartupGame-Klasse	108
Die main()-Methode des Spiels	110
Dann spielen wir mal …	113
Mehr über for-Schleifen	114
Die verbesserte for-Schleife	116
Elementare Typen casten	117
Übungen	118

6 Die Java-Bibliothek verwenden

Java wird mit Hunderten vorgefertigter Klassen ausgeliefert. Sie müssen das Rad nicht neu erfinden, wenn Sie wissen, wie Sie das Benötigte in der Java-Bibliothek, der sogenannten **Java-API**, finden können. *Sie haben Besseres zu tun.* Wenn Sie Code schreiben, reicht es völlig, wenn Sie sich *nur auf die Teile* konzentrieren, die bei Ihrer Anwendung besonders sind. Die Java-Kernbibliothek ist ein gigantischer Haufen Klassen, die nur darauf warten, dass Sie sie als Bausteine für Ihre eigenen Programme verwenden.

»Gut zu wissen, dass das java.util-Package eine ArrayList-Klasse bereitstellt. Selbst hätte ich das aber nicht herausgefunden.«

- Julia, 31, Hand-Model

Das vorige Kapitel endete mit einem Cliffhanger – nämlich einem Bug.	126
Wachen Sie endlich auf und sehen Sie der Java-API ins Auge	132
Einige Dinge, die Sie mit ArrayList tun können	133
ArrayList mit einem regulären Array vergleichen	137
Bauen wir das RICHTIGE Spiel: »Startups versenken«	140
Vorcode für die echte StartupBust-Klasse	144
Die finale Version der Startup-Klasse	150
Boolesche Ausdrücke mit Superkräften	151
Die Bibliothek (Java-API) verwenden	154
Übungen	163

7 Besser leben in Objectville

Planen Sie Ihre Programme mit der Zukunft im Blick. Was wäre, wenn Sie Code schreiben könnten, den ein *anderer* **problemlos** erweitern kann? Und was, wenn Ihr Code flexibel genug für diese lästigen Änderungen der Spezifikation in letzter Minute wäre? Wenn Sie auf den Polymorphie-Zug springen, lernen Sie die fünf Schritte zu besserem Klassendesign, 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.

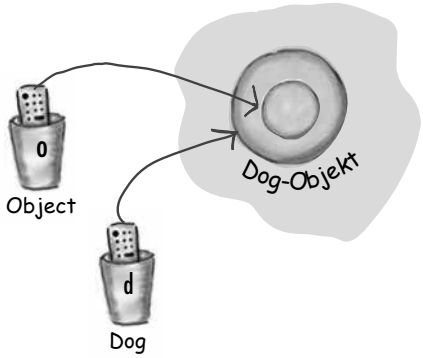
Stuhlkriege neu aufgelegt ...	168
Vererbung verstehen	170
Entwerfen wir einen Vererbungsbaum für ein Tiersimulationsprogramm	172
Weitere Vererbungsmöglichkeiten finden	175
IST EIN und HAT EIN verwenden	179
Woher wissen Sie, dass Ihre Vererbungshierarchie korrekt ist?	181
Vererbung kann man gut gebrauchen, man kann sie aber auch missbrauchen!	183
Den Vertrag einhalten: Regeln für das Überschreiben	192
Eine Methode überladen	193
Übungen	194

8 Ernsthafte Polymorphie

Vererbung ist nur der Anfang. Um Polymorphie nutzen zu können, benötigen wir Interfaces. 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 zu 100 % abstrakte Klasse. Was eine abstrakte Klasse ist? Das ist eine Klasse, die nicht instanziiert werden kann. Und wozu soll die gut sein? Lesen Sie das Kapitel ...

```
Object o = al.get(index);
Dog d = (Dog) o;
d.roam();
```

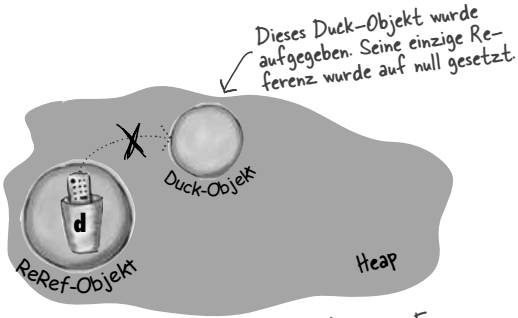
Das Object wieder zu dem Dog casten, das es die ganze Zeit war.



Haben wir etwas vergessen, als wir das hier entworfen haben?	200
Der Compiler verhindert die Instanziierung von abstrakten Klassen	203
Abstrakt vs. konkret	204
Abstrakte Methoden MÜSSEN implementiert werden	206
Polymorphie in Aktion	208
Was ist mit Nicht-Animals? Warum machen wir die Klasse nicht so allgemein, dass sie mit allem umgehen kann?	210
Wenn ein Dog sich nicht wie ein Dog verhält	214
Erforschen wir ein paar Entwurfsoptionen	221
Das Pet-Interface erstellen und implementieren	227
Die Superklassenversion einer Methode aufrufen	230
Übungen	232

9 Leben und Sterben eines Objekts

Objekte werden geboren und Objekte sterben. Sie herrschen über das Leben eines Objekts. Sie entscheiden, wie und wann es *konstruiert* wird. Sie entscheiden, wann es zerstört wird. Der **Garbage Collector (gc)** will den Speicher wiederhaben. Wir sehen uns an, wie Objekte erzeugt werden, wo sie leben und wie man sie effizient gehen lässt. Das bedeutet, dass wir über den Heap, den Stack, Geltungsbereiche, Konstruktoren, Superklassenkonstruktoren, Nullreferenzen und mehr reden werden.



Der Stack und der Heap: wo das Leben spielt	238
Methoden werden gestapelt	239
Was ist mit lokalen Variablen, die Objekte sind?	240
Das Wunder der Objekterstellung	242
Ein Duck-Objekt konstruieren	244
Erstellt der Compiler nicht immer einen argumentlosen Konstruktor für Sie?	248
Kurzer Rückblick. Vier Dinge, die Sie sich zu Konstruktoren merken sollten!	251
Die Rolle der Superklassenkonstruktoren im Leben eines Objekts	253
Kann ein Kind vor seinen Eltern existieren?	256
Was ist mit Referenzvariablen?	262
Mir gefällt nicht, worauf das hinausläuft.	263
Übungen	268

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 Tausenderpunkten unterteilt werden. Und wie steht es mit Datumsangaben? Und wenn Sie einen String in eine Zahl parsen wollen? Oder die Zahl in einen String? Als Erstes sehen wir uns an, was es für eine Variable oder eine Methode bedeutet, statisch zu sein.

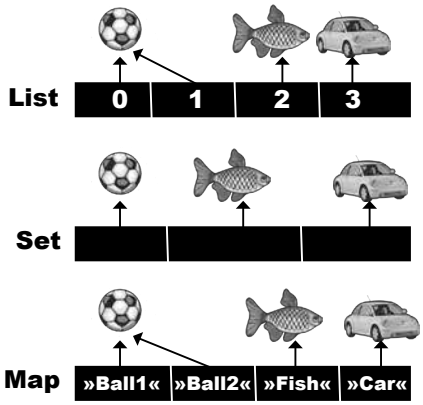
Statische Variablen werden geteilt.

Alle Instanzen derselben Klasse teilen sich eine einzelne Kopie der statischen Variablen.

MATH-Methoden: Näher werden Sie einer globalen Methode nie wieder kommen	276
Der Unterschied zwischen regulären (nicht statischen) und statischen Methoden	277
Eine statische Variable initialisieren	283
Math-Methoden	288
Elementartypen verpacken	290
Autoboxing funktioniert fast überall	292
Und umgekehrt ... eine elementare Zahl in einen String verwandeln	295
Zahlenformatierung	296
Der Format-Spezifizierer	300
Übungen	306

11 Datenstrukturen

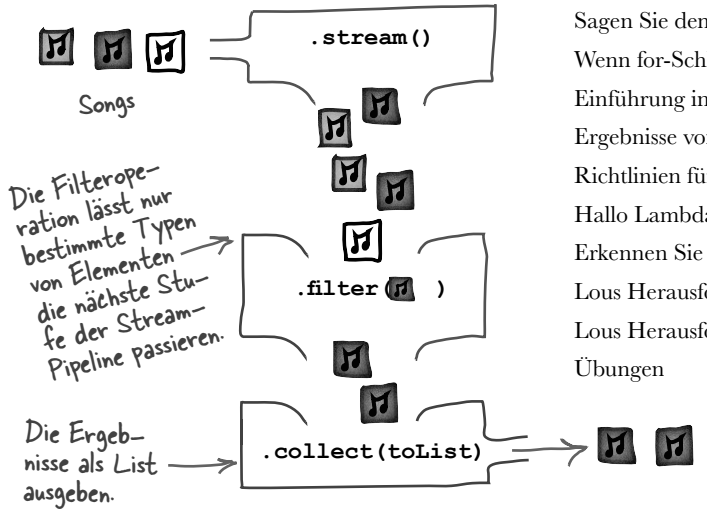
Sortieren ist in Java ein Klacks. Ihnen stehen alle Werkzeuge für die Sammlung und Verarbeitung von Daten zur Verfügung, ohne dass Sie Ihren eigenen Sortieralgorithmus schreiben müssen. Im Java-Collections-Framework finden Sie Datenstrukturen für quasi jeden Zweck, den Sie sich nur vorstellen können. Brauchen Sie eine Liste, die einfach um neue Elemente erweitert werden kann? Wollen Sie etwas anhand seines Namens finden? Möchten Sie eine Liste erstellen, die automatisch Duplikate entfernt? Sie möchten Ihre Kollegen danach sortieren, wie oft sie Ihnen in den Rücken gefallen sind? Es ist alles da ...



Die java.util-API, List und Collections entdecken	314
Generics sorgen für mehr Typsicherheit	320
Erneut ein Blick auf die sort()-Methode	327
Die neue, verbesserte Song-Klasse	330
Mit Comparators sortieren	336
Die Jukebox mit Lambdas aktualisieren	342
Ein HashSet anstelle einer ArrayList verwenden	347
Was Sie über TreeSet wissen MÜSSEN ...	353
Nachdem wir Lists und Sets gesehen haben, verwenden wir jetzt eine Map	355
Endlich wieder zurück zu Generics	358
Lösung zur Übung	364

12 Lambdas und Streams: Was – nicht wie!

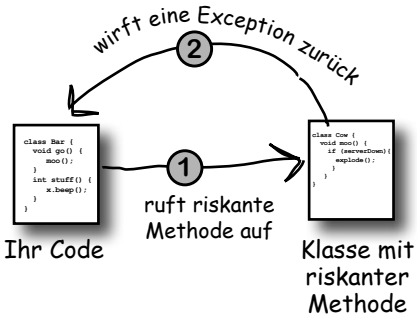
Was wäre, wenn ... Sie dem Computer nicht mitteilen müssten, WIE er etwas tun soll? In diesem Kapitel befassen wir uns mit der Streams-API, und Sie werden sehen, wie nützlich Lambda-Ausdrücke bei der Verwendung von Streams sein können. Außerdem werden Sie lernen, wie Sie mithilfe der Streams-API die Daten in einer Collection abfragen und umwandeln können.



Sagen Sie dem Computer, WAS Sie wollen	370
Wenn for-Schleifen schieflaufen	372
Einführung in die Streams-API	375
Ergebnisse von einem Stream erhalten	378
Richtlinien für die Arbeit mit Streams	384
Hallo Lambda, mein (nicht ganz so) alter Freund	388
Erkennen Sie funktionale Interfaces	396
Lous Herausforderung Nr. 1: Finde alle »Rock«-Songs	400
Lous Herausforderung Nr. 2: Alle Genres auflisten	404
Übungen	415

13 Riskantes Verhalten

Dinge passieren. Die Datei fehlt. Der Server ist down. Auch 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 mit den schlimmen Dingen umgeht, die passieren könnten. 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 eine Musikmaschine	422
Zuerst brauchen wir einen Sequencer	424
Eine Exception ist ein Objekt ... des Typs Exception	428
Flusskontrolle in try/catch-Blöcken	432
Eine Methode kann mehr als eine Exception werfen	435
Mehrfache catch-Blöcke müssen von klein nach groß geordnet werden	438
Ausweichen (durch Deklaration) verzögert nur das Unausweichliche	442
Codeküche	445
Version 1: Ihre allererste Soundplayer-App	448
Version 2: Kommandozeilenargumente für das Experimentieren mit Sounds	452
Übungen	454

14 Innen hui, außen GUI

Sehen Sie den Tatsachen ins Auge: Früher oder später werden Sie GUIs erstellen müssen. Selbst wenn Sie davon ausgehen, dass Sie den Rest Ihres Lebens serverseitigen Code schreiben, werden Sie irgendwann Werkzeuge schreiben müssen, die ein GUI benötigen. Wir werden zwei Kapitel mit der Programmierung von GUIs zubringen und dabei grundlegende Features der Sprache Java kennenlernen, z. B. **Event-Handling** und **innere Klassen**. Wir stellen einen Button auf dem Bildschirm dar, zeigen ein JPEG-Bild an und versuchen uns sogar an einer kleinen Animation.

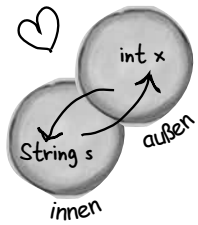
```
class MyOuter  {

    class MyInner {
       void go() {
         }
    }

}
```

Das äußere und das innere Objekt gehen eine intime Bindung ein.

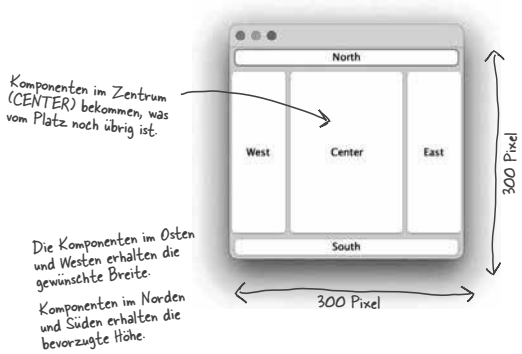
Diese beiden Objekte auf dem Heap haben eine besondere Bindung zueinander. Das innere Objekt kann die Variablen des äußeren Objekts benutzen (und umgekehrt).



Alles beginnt mit einem Fenster	462
Wie man an ein Benutzer-Event kommt	465
Listener, Quellen und Events	469
Machen Sie sich Ihr eigenes Grafik-Widget	472
Spaß mit paintComponent()	473
GUI-Layouts: mehrere Widgets in einen Frame packen	478
Die Rettung: Innere Klassen!	484
Rettung durch Lambdas!	490
Innere Klassen für Animationen einsetzen	492
Eine einfachere Möglichkeit, Messages und Events zu erstellen	498
Übungen	502

15 Mehr Schwung mit Swing

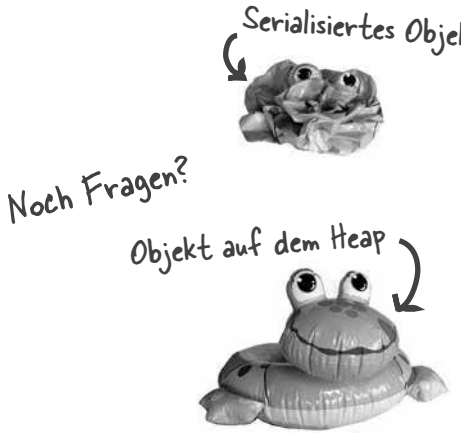
Swing ist einfach. Es sei denn, es ist Ihnen tatsächlich *wichtig*, wo die Dinge auf dem Bildschirm landen. Swing-Code *sieht* einfach aus, bis Sie ihn kompilieren und laufen lassen und denken: »Moment mal, *das* sollte aber *woandershin*!« Der Grund dafür, dass er *einfach* zu *programmieren*, aber *schwer* zu *kontrollieren* ist, ist der **Layoutmanager**. 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.



Swing-Komponenten	510
Layoutmanager	511
Die drei wichtigsten Layoutmanager: Border, Flow und Box	513
Es gibt keine Dummen Fragen	522
Spielen mit Swing-Komponenten	523
Codeküche	526
Die BeatBox	529
Übungen	534

16 Objekte (und Text) speichern

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 entweder *auf die harte Tour machen*, indem Sie jedes Objekt befragen und dann mühsam die Werte aller Instanzvariablen notieren. Oder **Sie gehen einfach nach OO-Art vor** – indem Sie lediglich das Objekt selbst gefriertrocknen (serialisieren) und deserialisieren, wenn Sie es zurückhaben wollen.



Ein serialisiertes Objekt in eine Datei schreiben	542
Soll eine Klasse serialisierbar sein, implementieren Sie	547
Deserialisierung: ein Objekt wiederherstellen	551
Version-ID: Ein großes Serialisierungsproblem	556
Einen String in ein Textdatei schreiben	559
Aus einer Textdatei lesen	566
Quiz Card Player (Code grob skizziert)	567
Path, Paths und Files (mit Verzeichnissen spielen)	573
Endlich, ein genauer Blick auf finally	574
Ein BeatBox-Pattern speichern	579
Übungen	580

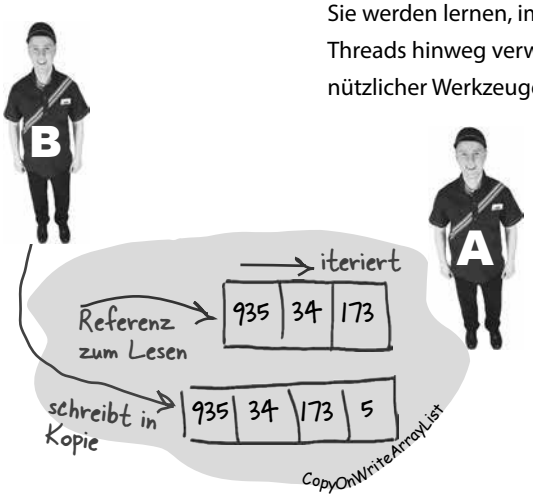
17 Eine Verbindung herstellen

Verbindung zur Außenwelt aufnehmen. Das ist nicht schwer. Um alle grundlegenden Netzwerkdetails kümmern sich Klassen in der java.net-Bibliothek. Einer der großen Vorteile von Java ist, dass das Senden und Empfangen über ein Netzwerk einfache Eingabe/Ausgabe ist, lediglich mit einem etwas anderen Anschluss-Stream am Ende der Kette. In diesem Kapitel erstellen wir *Client*-und *Server*-Sockets und *Clients* und *Server*. Bevor Sie mit dem Kapitel durch sind, haben Sie einen voll funktionsfähigen und Multithread-tauglichen Chatclient. Haben wir da gerade *Multithreading* erwähnt?

Verbinden, senden, empfangen	590
Der DailyAdviceClient (»Tipp des Tages«)	598
Ein einfaches Serverprogramm schreiben	601
Java hat mehrere Threads, aber nur eine Thread-Klasse	610
Die drei Zustände eines neuen Threads	616
Einen Thread schlafen schicken	622
Zwei (oder mehr!) Threads erzeugen und starten	626
Feierabend am Thread-Pool	629
Der neue und verbesserte SimpleChatClient	632
Übungen	634

18 Nebenläufigkeitsprobleme behandeln

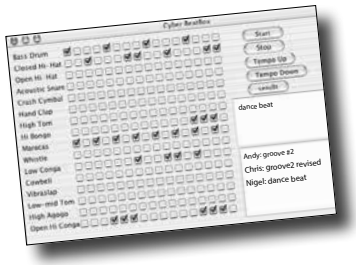
Es ist schwer, mehrere Dinge gleichzeitig zu tun. Multithreading-Code zu schreiben, ist einfach. Multithreading-Code zu schreiben, der wie erwartet funktioniert, kann viel schwieriger sein. In diesem letzten Kapitel zeigen wir Ihnen ein paar Dinge, die schiefgehen können, wenn mehrere Threads gleichzeitig arbeiten. Sie werden einige Werkzeuge in java.util.concurrent kennenlernen, die Ihnen helfen können, Multithreading-Code zu schreiben, der korrekt funktioniert. Sie werden lernen, immutable (unveränderliche) Objekte zu erstellen, die sicher über mehrere Threads hinweg verwendet werden können. Am Ende dieses Kapitels verfügen Sie über eine Reihe nützlicher Werkzeuge für die Arbeit mit Nebenläufigkeit.



Das Ryan-und-Monica-Problem, in Code ausgedrückt	642
Das Schloss eines Objekts verwenden	647
Das gefürchtete »Problem der verlorenen Aktualisierung«	650
Die increment()-Methode atomar machen. Synchronisieren Sie sie!	652
Blockade, eine tödliche Seite der Synchronisierung	654
»Vergleichen und tauschen« mit atomaren Variablen	656
Immutable Objekte verwenden	659
Mehr Probleme mit geteilten Daten	662
Verwenden Sie eine threadsichere Datenstruktur	664
Übungen	668

A Anhang A

Die finale Codeküche. Der vollständige Code für unser BeatBox-Clientprogramm – Ihre Chance, ein Rockstar zu werden!



Das endgültige BeatBox-Clientprogramm 674
Das endgültige BeatBox-Serverprogramm 681

B Anhang B

Die (mehr als) zehn wichtigsten 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.

#11 JShell (Java REPL) 684
#10 Packages 685
#9 Immutabilität in Strings und Wrappern 688
#8 Zugriffsebenen und Zugriffsmodifier (wer sieht was) 689
#7 Varargs 691
#6 Annotationen 692
#5 Lambdas und Maps 693
#4 Parallele Streams 695
#3 Enumerations (enumerierte Typen/Enums) 696
#2 Lokale Typinferenz für Variablen (var) 698
#1 Records 699

i Index

701