

Your Brain on Java—A Learner's Guide

2nd Edition - Covers Java 5.0

Head First Java

Learn how threads
can change your life



Make Java concepts
stick to your brain



Avoid embarrassing
OO mistakes

Fool around in
the Java Library



Bend your mind
around 42
Java puzzles



Make attractive
and useful GUIs



O'REILLY®

Kathy Sierra & Bert Bates

Table of Contents (summary)

	Intro	xxi
1	Breaking the Surface: <i>a quick dip</i>	1
2	A Trip to Objectville: <i>yes, there will be objects</i>	27
3	Know Your Variables: <i>primitives and references</i>	49
4	How Objects Behave: <i>object state affects method behavior</i>	71
5	Extra-Strength Methods: <i>flow control, operations, and more</i>	95
6	Using the Java Library: <i>so you don't have to write it all yourself</i>	125
7	Better Living in Objectville: <i>planning for the future</i>	165
8	Serious Polymorphism: <i>exploiting abstract classes and interfaces</i>	197
9	Life and Death of an Object: <i>constructors and memory management</i>	235
10	Numbers Matter: <i>math, formatting, wrappers, and statics</i>	273
11	Risky Behavior: <i>exception handling</i>	315
12	A Very Graphic Story: <i>intro to GUI, event handling, and inner classes</i>	353
13	Work on Your Swing: <i>layout managers and components</i>	399
14	Saving Objects: <i>serialization and I/O</i>	429
15	Make a Connection: <i>networking sockets and multithreading</i>	471
16	Data Structures: <i>collections and generics</i>	529
17	Release Your Code: <i>packaging and deployment</i>	581
18	Distributed Computing: <i>RMI with a dash of servlets, EJB, and Jini</i>	607
A	Appendix A: <i>Final code kitchen</i>	649
B	Appendix B: <i>Top Ten Things that didn't make it into the rest of the book</i>	659
	Index	677

Table of Contents (the full version)



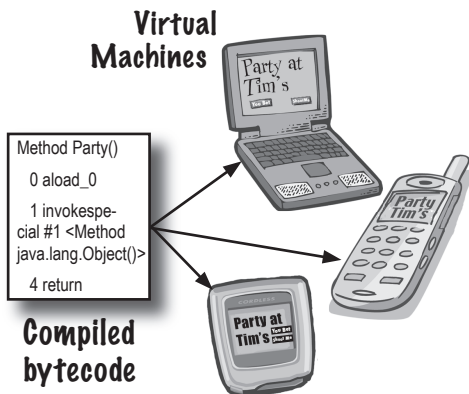
Intro

Your brain on Java. Here you are trying to *learn* something, while here your *brain* is doing you a favor by making sure the learning doesn't *stick*. Your brain's thinking, "Better leave room for more important things, like which wild animals to avoid and whether naked snowboarding is a bad idea." So how *do* you trick your brain into thinking that your life depends on knowing Java?

Who is this book for?	xxii
What your brain is thinking	xxiii
Metacognition	xxv
Bend your brain into submission	xxvii
What you need for this book	xxviii
Technical editors	xxx
Acknowledgements	xxxi

1 Breaking the Surface

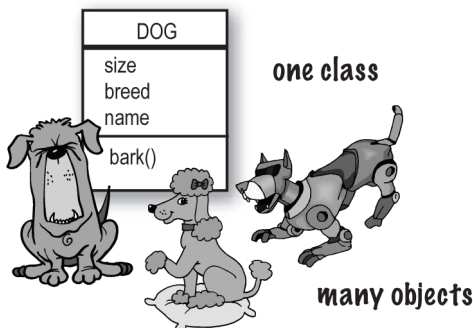
Java takes you to new places. From its humble release to the public as the (wimpy) version 1.02, Java seduced programmers with its friendly syntax, object-oriented features, memory management, and best of all—the promise of portability. We'll take a quick dip and write some code, compile it, and run it. We're talking syntax, loops, branching, and what makes Java so cool. Dive in.



The way Java works	2
Code structure in Java	7
Anatomy of a class	8
The main() method	9
Looping	11
Conditional branching (<i>if</i> tests)	13
Coding the “99 bottles of beer” app	14
Phrase-o-matic	16
Fireside chat: compiler vs. JVM	18
Exercises and puzzles	20

2 A Trip to Objectville

I was told there would be objects. In Chapter 1, we put all of our code in the main() method. That's not exactly object-oriented. So now we've got to leave that procedural world behind and start making some objects of our own. We'll look at what makes object-oriented (OO) development in Java so much fun. We'll look at the difference between a class and an object. We'll look at how objects can improve your life.

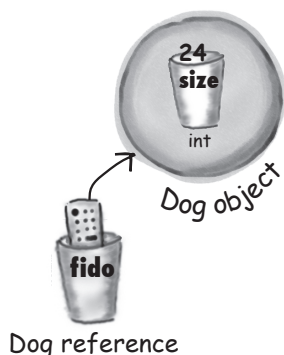


Chair Wars (Brad the OO guy vs. Larry the procedural guy)	28
Inheritance (an introduction)	31
Overriding methods (an introduction)	32
What's in a class? (methods, instance variables)	34
Making your first object	36
Using main()	38
Guessing Game code	39
Exercises and puzzles	42

3 Know Your Variables

Variables come in two flavors: primitive and reference.

There's gotta be more to life than integers, Strings, and arrays. What if you have a PetOwner object with a Dog instance variable? Or a Car with an Engine? In this chapter we'll unwrap the mysteries of Java types and look at what you can *declare* as a variable, what you can *put* in a variable, and what you can *do* with a variable. And we'll finally see what life is truly like on the garbage-collectible heap.

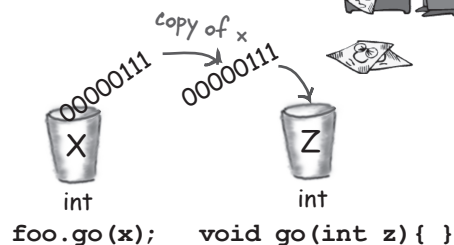


Declaring a variable (Java cares about <i>type</i>)	50
Primitive types ("I'd like a double with extra foam, please")	51
Java keywords	53
Reference variables (remote control to an object)	54
Object declaration and assignment	55
Objects on the garbage-collectible heap	57
Arrays (a first look)	59
Exercises and puzzles	63

4 How Objects Behave

State affects behavior, behavior affects state. We know that objects have **state** and **behavior**, represented by **instance variables** and **methods**. Now we'll look at how state and behavior are *related*. An object's behavior uses an object's unique state. In other words, **methods use instance variable values**. Like, "if dog weight is less than 14 pounds, make yippy sound, else..." **Let's go change some state!**

**pass-by-value means
pass-by-copy**



Methods use object state (bark different)	73
Method arguments and return types	74
Pass-by-value (the variable is <i>always</i> copied)	77
Getters and Setters	79
Encapsulation (do it or risk humiliation)	80
Using references in an array	83
Exercises and puzzles	88

5 Extra-Strength Methods

Let's put some muscle in our methods. You dabbled with variables, played with a few objects, and wrote a little code. But you need more tools. Like **operators**. And **loops**. Might be useful to **generate random numbers**. And **turn a String into an int**, yeah, that would be cool. And why don't we learn it all by *building* something real, to see what it's like to write (and test) a program from scratch. **Maybe a game**, like Sink a Dot Com (similar to Battleship).

We're gonna build the Sink a Dot Com game

A							
B	Go2.com						
C	Go2.com						
D	Go2.com		Pets.com				
E							
F							
G				AskMe.com			
	0	1	2	3	4	5	6

Building the Sink a Dot Com game	96
Starting with the Simple Dot Com game (a simpler version)	98
Writing precode (pseudocode for the game)	100
Test code for Simple Dot Com	102
Coding the Simple Dot Com game	103
Final code for Simple Dot Com	106
Generating random numbers with Math.random()	111
Ready-bake code for getting user input from the command-line	112
Looping with <i>for</i> loops	114
Casting primitives from a large size to a smaller size	117
Converting a String to an int with Integer.parseInt()	117
Exercises and puzzles	118

6 Using the Java Library

Java ships with hundreds of pre-built classes. You don't have to reinvent the wheel if you know how to find what you need from the Java library, commonly known as the **Java API**. *You've got better things to do.* If you're going to write code, you might as well write *only* the parts that are custom for your application. The core Java library is a giant pile of classes just waiting for you to use like building blocks.

"Good to know there's an ArrayList in the java.util package. But by myself, how would I have figured that out?"

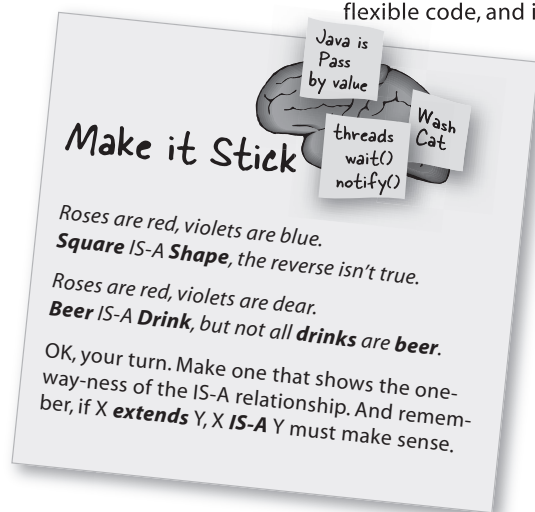
- Julia, 31, hand model



Analyzing the bug in the Simple Dot Com Game	126
ArrayList (taking advantage of the Java API)	132
Fixing the DotCom class code	138
Building the <i>real</i> game (Sink a Dot Com)	140
Precode for the <i>real</i> game	144
Code for the <i>real</i> game	146
<i>boolean</i> expressions	151
Using the library (Java API)	154
Using packages (import statements, fully-qualified names)	155
Using the HTML API docs and reference books	158
Exercises and puzzles	161

7 Better Living in Objectville

Plan your programs with the future in mind. What if you could write code that someone *else* could extend, **easily**? What if you could write code that was flexible, for those pesky last-minute spec changes? When you get on the Polymorphism Plan, you'll learn the 5 steps to better class design, the 3 tricks to polymorphism, the 8 ways to make flexible code, and if you act now—a bonus lesson on the 4 tips for exploiting inheritance.

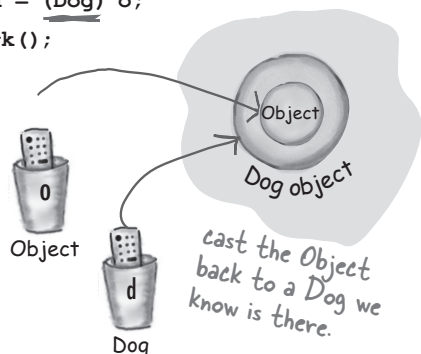


Understanding inheritance (superclass and subclass relationships)	168
Designing an inheritance tree (the Animal simulation)	170
Avoiding duplicate code (using inheritance)	171
Overriding methods	172
IS-A and HAS-A (bathtub girl)	177
What do you inherit from your superclass?	180
What does inheritance really <i>buy</i> you?	182
Polymorphism (using a supertype reference to a subclass object)	183
Rules for overriding (don't touch those arguments and return types!)	190
Method overloading (nothing more than method name re-use)	191
Exercises and puzzles	192

8 Serious Polymorphism

Inheritance is just the beginning. To exploit polymorphism, we need interfaces. We need to go beyond simple inheritance to flexibility you can get only by designing and coding to interfaces. What's an interface? A 100% abstract class. What's an abstract class? A class that can't be instantiated. What's that good for? Read the chapter...

```
Object o = al.get(id);
Dog d = (Dog) o;
d.bark();
```

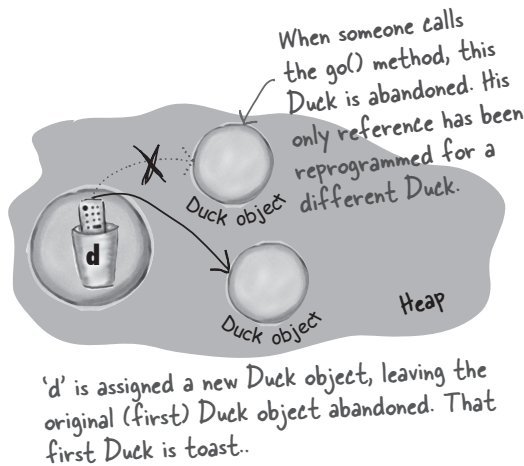


Some classes just should <i>not</i> be instantiated	200
Abstract classes (<i>can't</i> be instantiated)	201
Abstract methods (must be implemented)	203
Polymorphism in action	206
Class Object (the ultimate superclass of <i>everything</i>)	208
Taking objects out of an ArrayList (they come out as type Object)	211
Compiler checks the reference type (before letting you call a method)	213
Get in touch with your inner object	214
Polymorphic references	215
Casting an object reference (moving lower on the inheritance tree)	216
Deadly Diamond of Death (multiple inheritance problem)	223
Using interfaces (the best solution!)	224
Exercises and puzzles	230



9 Life and Death of an Object

Objects are born and objects die. You're in charge. You decide when and how to *construct* them. You decide when to *abandon* them. The **Garbage Collector (gc)** reclaims the memory. We'll look at how objects are created, where they live, and how to keep or abandon them efficiently. That means we'll talk about the heap, the stack, scope, constructors, super constructors, null references, and gc eligibility.

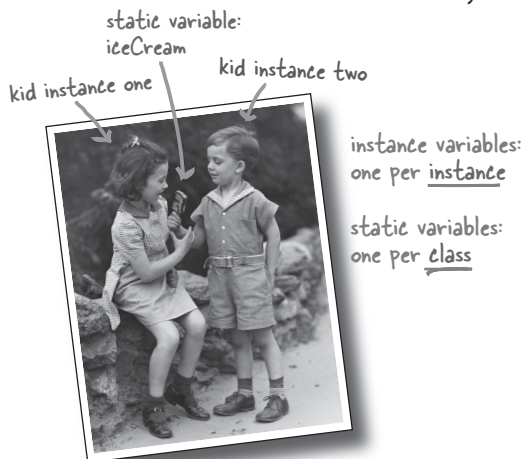


The stack and the heap, where objects and variables live	236
Methods on the stack	237
Where <i>local</i> variables live	238
Where <i>instance</i> variables live	239
The miracle of object creation	240
Constructors (the code that runs when you say <i>new</i>)	241
Initializing the state of a new Duck	243
Overloaded constructors	247
Superclass constructors (constructor chaining)	250
Invoking overloaded constructors using <i>this()</i>	256
Life of an object	258
Garbage Collection (and making objects eligible)	260
Exercises and puzzles	266

10 Numbers Matter

Static variables are shared by all instances of a class.

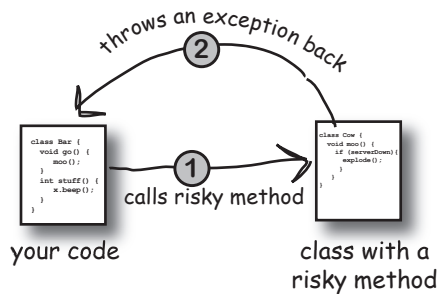
Do the Math. The Java API has methods for absolute value, rounding, min/max, etc. But what about formatting? You might want numbers to print exactly two decimal points, or with commas in all the right places. And you might want to print and manipulate dates, too. And what about parsing a String into a number? Or turning a number into a String? We'll start by learning what it means for a variable or method to be *static*.



Math class (do you really need an instance of it?)	274
static methods	275
static variables	277
Constants (static final variables)	282
Math methods (random(), round(), abs(), etc.)	286
Wrapper classes (Integer, Boolean, Character, etc.)	287
Autoboxing	289
Number formatting	294
Date formatting and manipulation	301
Static imports	307
Exercises and puzzles	310

11 Risky Behavior

Stuff happens. The file isn't there. The server is down. No matter how good a programmer you are, you can't control *everything*. When you write a risky method, you need code to handle the bad things that might happen. But how do you *know* when a method is risky? Where do you put the code to *handle* the **exceptional** situation? In *this* chapter, we're going to build a MIDI Music Player, that uses the risky JavaSound API, so we better find out.



Making a music machine (the BeatBox)	316
What if you need to call risky code?	319
Exceptions say “something bad may have happened...”	320
The compiler guarantees (it <i>checks</i>) that you’re aware of the risks	321
Catching exceptions using a <i>try/catch</i> (skateboarder)	322
Flow control in <i>try/catch</i> blocks	326
The <i>finally</i> block (no matter what happens, turn off the oven!)	327
Catching multiple exceptions (the order matters)	329
Declaring an exception (just duck it)	335
Handle or declare law	337
Code Kitchen (making sounds)	339
Exercises and puzzles	348

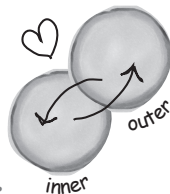
12 A Very Graphic Story

Face it, you need to make GUIs. Even if you believe that for the rest of your life you'll write only server-side code, sooner or later you'll need to write tools, and you'll want a graphical interface. We'll spend two chapters on GUIs, and learn more language features including **Event Handling** and **Inner Classes**. We'll put a button on the screen, we'll paint on the screen, we'll display a jpeg image, and we'll even do some animation.

```
class MyOuter {  
    class MyInner {  
        void go() {  
        }  
    }  
}
```

The outer and inner objects are now intimately linked.

These two objects on the heap have a special bond. The inner can use the outer's variables (and vice-versa).



Your first GUI	355
Getting a user event	357
Implement a listener interface	358
Getting a button's <code>ActionEvent</code>	360
Putting graphics on a GUI	363
Fun with <code>paintComponent()</code>	365
The <code>Graphics2D</code> object	366
Putting more than one button on a screen	370
Inner classes to the rescue (make your listener an inner class)	376
Animation (move it, paint it, move it, paint it, move it, paint it...)	382
Code Kitchen (painting graphics with the beat of the music)	386
Exercises and puzzles	394

13 Work on your Swing

Swing is easy. Unless you actually *care* where everything goes. Swing code *looks* easy, but then compile it, run it, look at it and think, “hey, *that’s* not supposed to go *there*.” The thing that makes it *easy* to *code* is the thing that makes it *hard* to *control*—the **Layout Manager**. But with a little work, you can get layout managers to submit to your will. In this chapter, we’ll work on our Swing and learn more about widgets.

Components in the east and west get their preferred width.

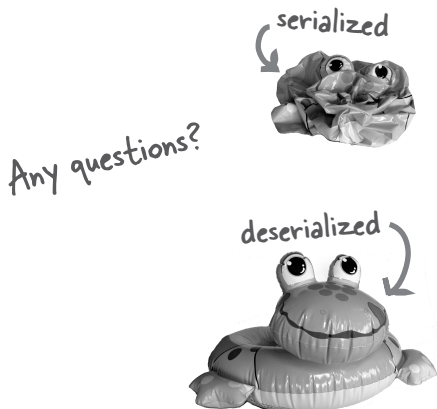
Things in the north and south get their preferred height.



Swing Components	400
Layout Managers (they control size and placement)	401
Three Layout Managers (border, flow, box)	403
BorderLayout (cares about five regions)	404
FlowLayout (cares about the order and preferred size)	408
BoxLayout (like flow, but can stack components vertically)	411
JTextField (for single-line user input)	413
JTextArea (for multi-line, scrolling text)	414
JCheckBox (is it selected?)	416
JList (a scrollable, selectable list)	417
Code Kitchen (The Big One - building the BeatBox chat client)	418
Exercises and puzzles	424

14 Saving Objects

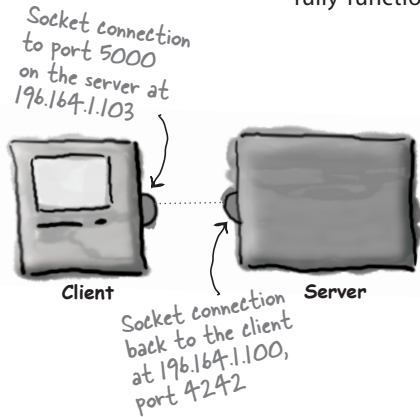
Objects can be flattened and inflated. Objects have state and behavior. Behavior lives in the class, but *state* lives within each individual *object*. If your program needs to save state, *you can do it the hard way*, interrogating each object, painstakingly writing the value of each instance variable. Or, **you can do it the easy OO way**—you simply freeze-dry the object (serialize it) and reconstitute (deserialize) it to get it back.



Saving object state	431
Writing a serialized object to a file	432
Java input and output streams (connections and chains)	433
Object serialization	434
Implementing the Serializable interface	437
Using transient variables	439
Deserializing an object	441
Writing to a text file	447
java.io.File	452
Reading from a text file	454
Splitting a String into tokens with split()	458
CodeKitchen	462
Exercises and puzzles	466

15 Make a Connection

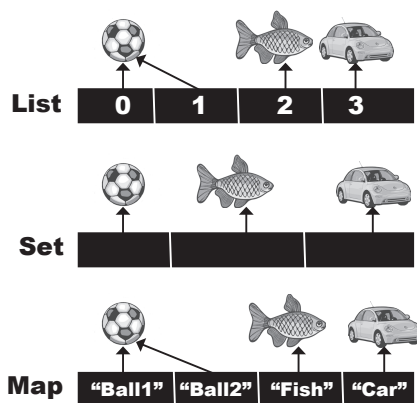
Connect with the outside world. It's easy. All the low-level networking details are taken care of by classes in the `java.net` library. One of Java's best features is that sending and receiving data over a network is really just I/O with a slightly different connection stream at the end of the chain. In this chapter we'll make client sockets. We'll make server sockets. We'll make clients and servers. Before the chapter's done, you'll have a fully-functional, multithreaded chat client. Did we just say *multithreaded*?



Chat program overview	473
Connecting, sending, and receiving	474
Network sockets	475
TCP ports	476
Reading data from a socket (using <code>BufferedReader</code>)	478
Writing data to a socket (using <code>PrintWriter</code>)	479
Writing the Daily Advice Client program	480
Writing a simple server	483
Daily Advice Server code	484
Writing a chat client	486
Multiple call stacks	490
Launching a new thread (make it, start it)	492
The <code>Runnable</code> interface (the thread's job)	494
Three states of a new <code>Thread</code> object (new, runnable, running)	495
The runnable-running loop	496
Thread scheduler (it's his decision, not yours)	497
Putting a thread to sleep	501
Making and starting two threads	503
Concurrency issues: can this couple be saved?	505
The Ryan and Monica concurrency problem, in code	506
Locking to make things atomic	510
Every object has a lock	511
The dreaded "Lost Update" problem	512
Synchronized methods (using a lock)	514
Deadlock!	516
Multithreaded <code>ChatClient</code> code	518
Ready-bake <code>SimpleChatServer</code>	520
Exercises and puzzles	524

16 Data Structures

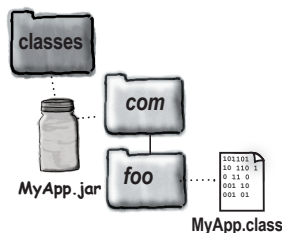
Sorting is a snap in Java. You have all the tools for collecting and manipulating your data without having to write your own sort algorithms. The Java Collections Framework has a data structure that should work for virtually anything you'll ever need to do. Want to keep a list that you can easily keep adding to? Want to find something by name? Want to create a list that automatically takes out all the duplicates? Sort your co-workers by the number of times they've stabbed you in the back?



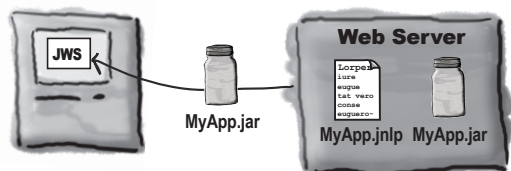
Collections	533
Sorting an ArrayList with Collections.sort()	534
Generics and type-safety	540
Sorting things that implement the Comparable interface	547
Sorting things with a custom Comparator	552
The collection API—lists, sets, and maps	557
Avoiding duplicates with HashSet	559
Overriding hashCode() and equals()	560
HashMap	567
Using wildcards for polymorphism	574
Exercises and puzzles	576

17 Release Your Code

It's time to let go. You wrote your code. You tested your code. You refined your code. You told everyone you know that if you never saw a line of code again, that'd be fine. But in the end, you've created a work of art. The thing actually runs! But now what? In these final two chapters, we'll explore how to organize, package, and deploy your Java code. We'll look at local, semi-local, and remote deployment options including executable jars, Java Web Start, RMI, and Servlets. Relax. Some of the coolest things in Java are easier than you think.

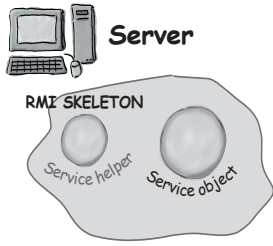


Deployment options	582
Keep your source code and class files separate	584
Making an executable JAR (Java ARchives)	585
Running an executable JAR	586
Put your classes in a package!	587
Packages must have a matching directory structure	589
Compiling and running with packages	590
Compiling with -d	591
Making an executable JAR (with packages)	592
Java Web Start (JWS) for deployment from the web	597
How to make and deploy a JWS application	600
Exercises and puzzles	601



18 Distributed Computing

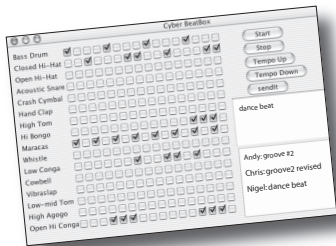
Being remote doesn't have to be a bad thing. Sure, things *are* easier when all the parts of your application are in one place, in one heap, with one JVM to rule them all. But that's not always possible. Or desirable. What if your application handles powerful computations? What if your app needs data from a secure database? In this chapter, we'll learn to use Java's amazingly simple Remote Method Invocation (RMI). We'll also take a quick peek at Servlets, Enterprise Java Beans (EJB) , and Jini.



Java Remote Method Invocation (RMI), hands-on, <i>very</i> detailed	614
Servlets (a quick look)	625
Enterprise JavaBeans (EJB), a <i>very</i> quick look	631
Jini, the best trick of all	632
Building the really cool universal service browser	636
The End	648

A Appendix A

The final Code Kitchen project. All the code for the full client-server chat beat box. Your chance to be a rock star.



BeatBoxFinal (client code)	650
MusicServer (server code)	657

B Appendix B

The Top Ten Things that didn't make it into the book. We can't send you out into the world just yet. We have a few more things for you, but this *is* the end of the book. And this time we really mean it.

Top Ten List	660
--------------	-----