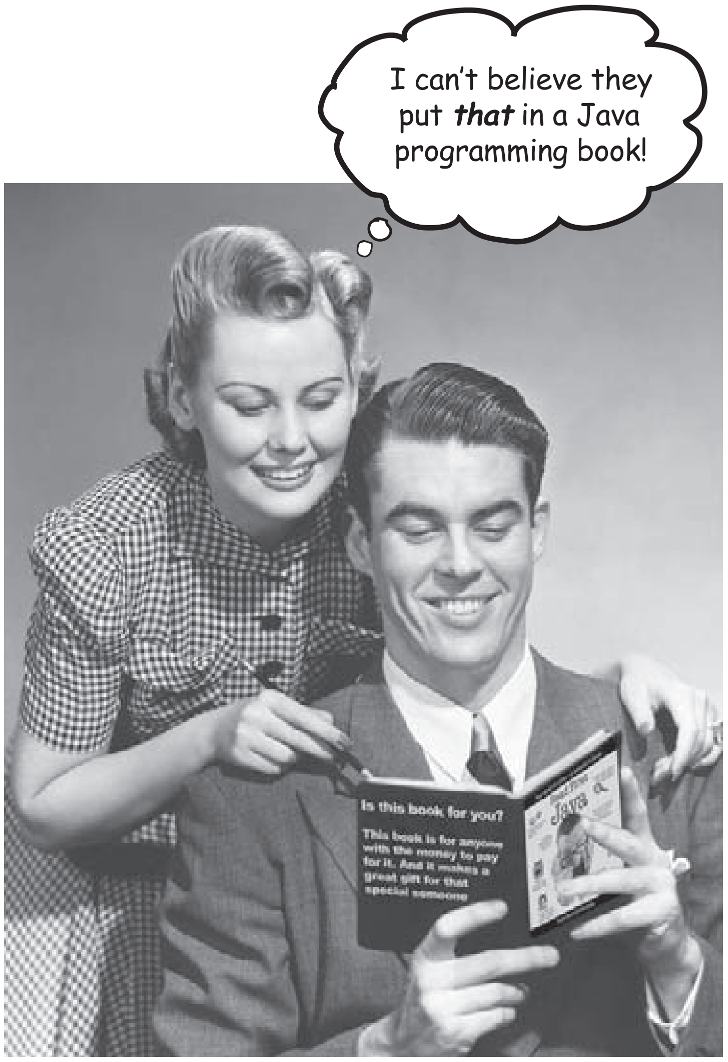
how to use this book

Intro


I can't believe they put ***that*** in a Java programming book!

Is this book for you?

This book is for anyone with the money to pay for it. And it makes a great gift for that special someone

In this section, we answer the burning question:
"So, why DID they put that in a Java programming book?"

Who is this book for?

If you can answer "yes" to ***all*** of these:

1. **Have you done some programming?**
2. **Do you want to learn Java?**
3. **Do you prefer stimulating dinner party conversation to dry, dull, technical lectures?**

this book is for you.

This is NOT a reference book. Head First Java is a book designed for *learning*, not an encyclopedia of Java facts.

Who should probably back away from this book?

If you can answer "yes" to any ***one*** of these:

1. **Is your programming background limited to HTML only, with no scripting language experience?**
   (If you've done anything with looping, or if/then logic, you'll do fine with this book, but HTML tagging alone might not be enough.)
2. **Are you a kick-butt C++ programmer looking for a *reference* book?**
3. **Are you afraid to try something different? Would you rather have a root canal than mix stripes with plaid? Do you believe than a technical book can't be serious if there's a picture of a duck in the memory management section?**

this book is ***not*** for you.

[note from marketing: who took out the part about how this book is for anyone with a valid credit card? And what about that "Give the Gift of Java" holiday promotion we discussed... -Fred]

We know what you're thinking.

"How can *this* be a serious Java programming book?"

"What's with all the graphics?"

"Can I actually *learn* it this way?"

"Do I smell pizza?"

And we know what your *brain* is thinking.

Your brain craves novelty. It's always searching, scanning, *waiting* for something unusual. It was built that way, and it helps you stay alive.

Today, you're less likely to be a tiger snack. But your brain's still looking. You just never know.

So what does your brain do with all the routine, ordinary, normal things you encounter? Everything it *can* to stop them from interfering with the brain's *real* job—recording things that *matter*. It doesn't bother saving the boring things; they never make it past the "this is obviously not important" filter.

How does your brain *know* what's important? Suppose you're out for a day hike and a tiger jumps in front of you, what happens inside your head?

Neurons fire. Emotions crank up. *Chemicals surge.*

And that's how your brain knows...

This must be important! Don't forget it!

But imagine you're at home, or in a library. It's a safe, warm, tiger-free zone. You're studying. Getting ready for an exam. Or trying to learn some tough technical topic your boss thinks will take a week, ten days at the most.

Great. Only 637 more dull, dry, boring pages.
your brain thinks THIS isn't worth saving.

Just one problem. Your brain's trying to do you a big favor. It's trying to make sure that this *obviously* non-important content doesn't clutter up scarce resources. Resources that are better spent storing the really *big* things. Like tigers. Like the danger of fire. Like how you should never again snowboard in shorts.

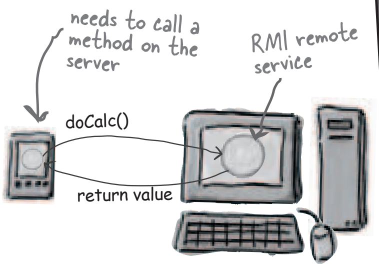
And there's no simple way to tell your brain, "Hey brain, thank you very much, but no matter how dull this book is, and how little I'm registering on the emotional richter scale right now, I really *do* want you to keep this stuff around."

We think of a "Head First Java" reader as a learner.

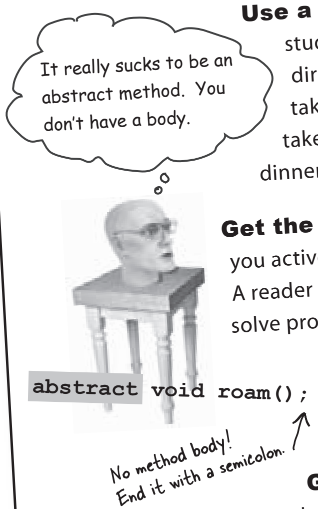
So what does it take to *learn* something? First, you have to *get* it, then make sure you don't *forget* it. It's not about pushing facts into your head. Based on the latest research in cognitive science, neurobiology, and educational psychology, *learning* takes a lot more than text on a page. We know what turns your brain on.

Some of the Head First learning principles:

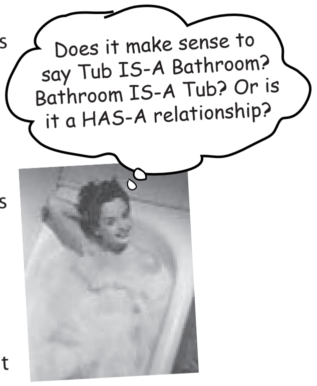
Make it visual. Images are far more memorable than words alone, and make learning much more effective (Up to 89% improvement in recall and transfer studies). It also makes things more understandable. **Put the words within or near the graphics** they relate to, rather than on the bottom or on another page, and learners will be up to *twice* as likely to solve problems related to the content.



Use a conversational and personalized style. In recent studies, students performed up to 40% better on post-learning tests if the content spoke directly to the reader, using a first-person, conversational style rather than taking a formal tone. Tell stories instead of lecturing. Use casual language. Don't take yourself too seriously. Which would *you* pay more attention to: a stimulating dinner party companion, or a lecture?


It really sucks to be an abstract method. You don't have a body.
abstract void roam();
No method body! End it with a semicolon.

Get the learner to think more deeply. In other words, unless you actively flex your neurons, nothing much happens in your head. A reader has to be motivated, engaged, curious, and inspired to solve problems, draw conclusions, and generate new knowledge. And for that, you need challenges, exercises, and thought-provoking questions, and activities that involve both sides of the brain, and multiple senses.


Does it make sense to say Tub IS-A Bathroom? Bathroom IS-A Tub? Or is it a HAS-A relationship?

Get—and keep—the reader's attention. We've all had the "I really want to learn this but I can't stay awake past page one" experience. Your brain pays attention to things that are out of the ordinary, interesting, strange, eye-catching, unexpected. Learning a new, tough, technical topic doesn't have to be boring. Your brain will learn much more quickly if it's not.

Touch their emotions. We now know that your ability to remember something is largely dependent on its emotional content. You remember what you care about. You remember when you feel something. No we're not talking heart-wrenching stories about a boy and his dog. We're talking emotions like surprise, curiosity, fun, "what the...?" , and the feeling of "I Rule!" that comes when you solve a puzzle, learn something everybody else thinks is hard, or realize you know something that "I'm more technical than thou" Bob from engineering *doesn't*.

Metacognition: thinking about thinking.

If you really want to learn, and you want to learn more quickly and more deeply, pay attention to how you pay attention. Think about how you think. Learn how you learn.

Most of us did not take courses on metacognition or learning theory when we were growing up. We were *expected* to learn, but rarely *taught* to learn.

But we assume that if you're holding this book, you want to learn Java. And you probably don't want to spend a lot of time.

To get the most from this book, or *any* book or learning experience, take responsibility for your brain. Your brain on *that* content.

The trick is to get your brain to see the new material you're learning as Really Important. Crucial to your well-being. As important as a tiger. Otherwise, you're in for a constant battle, with your brain doing its best to keep the new content from sticking.

I wonder how I can trick my brain into remembering this stuff...

So just how *DO* you get your brain to treat Java like it was a hungry tiger?

There's the slow, tedious way, or the faster, more effective way. The slow way is about sheer repetition. You obviously know that you *are* able to learn and remember even the dullest of topics, if you keep pounding on the same thing. With enough repetition, your brain says, "This doesn't *feel* important to him, but he keeps looking at the same thing *over* and *over* and *over*, so I suppose it must be."

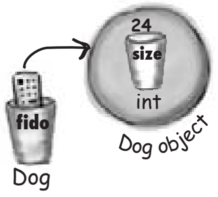
The faster way is to do ***anything that increases brain activity,*** especially different *types* of brain activity. The things on the previous page are a big part of the solution, and they're all things that have been proven to help your brain work in your favor. For example, studies show that putting words *within* the pictures they describe (as opposed to somewhere else in the page, like a caption or in the body text) causes your brain to try to makes sense of how the words and picture relate, and this causes more neurons to fire. More neurons firing = more chances for your brain to *get* that this is something worth paying attention to, and possibly recording.

A conversational style helps because people tend to pay more attention when they perceive that they're in a conversation, since they're expected to follow along and hold up their end. The amazing thing is, your brain doesn't necessarily *care* that the "conversation" is between you and a book! On the other hand, if the writing style is formal and dry, your brain perceives it the same way you experience being lectured to while sitting in a roomful of passive attendees. No need to stay awake.

But pictures and conversational style are just the beginning.

Here's what WE did:

We used ***pictures***, because your brain is tuned for visuals, not text. As far as your brain's concerned, a picture really *is* worth 1024 words. And when text and pictures work together, we embedded the text *in* the pictures because your brain works more effectively when the text is *within* the thing the text refers to, as opposed to in a caption or buried in the text somewhere.



We used ***repetition***, saying the same thing in different ways and with different media types, and *multiple senses*, to increase the chance that the content gets coded coded into more than one area of your brain.

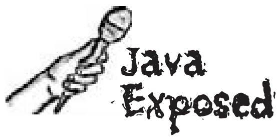
We used concepts and pictures in ***unexpected*** ways because your brain is tuned for novelty, and we used pictures and ideas with at least *some* ***emotional*** *content*, because your brain is tuned to pay attention to the biochemistry of emotions. That which causes you to *feel* something is more likely to be remembered, even if that feeling is nothing more than a little ***humor***, ***surprise***, or ***interest.***

We used a personalized, ***conversational style***, because your brain is tuned to pay more attention when it believes you're in a conversation than if it thinks you're passively listening to a presentation. Your brain does this even when you're *reading*.

We included more than 50 ***exercises*** , because your brain is tuned to learn and remember more when you ***do*** things than when you *read* about things. And we made the exercises challenging-yet-do-able, because that's what most *people* prefer.

We used ***multiple learning styles***, because *you* might prefer step-by-step procedures, while someone else wants to understand the big picture first, while someone else just wants to see a code example. But regardless of your own learning preference, *everyone* benefits from seeing the same content represented in multiple ways.

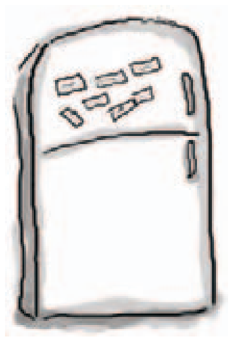
We include content for ***both sides of your brain***, because the more of your brain you engage, the more likely you are to learn and remember, and the longer you can stay focused. Since working one side of the brain often means giving the other side a chance to rest, you can be more productive at learning for a longer period of time.



And we included ***stories*** and exercises that present ***more than one point of view,*** because your brain is tuned to learn more deeply when it's forced to make evaluations and judgements.

We included ***challenges***, with exercises, and by asking ***questions*** that don't always have a straight answer, because your brain is tuned to learn and remember when it has to *work* at something (just as you can't get your *body* in shape by watching people at the gym). But we did our best to make sure that when you're working hard, it's on the *right* things. That ***you're not spending one extra dendrite*** processing a hard-to-understand example, or parsing difficult, jargon-laden, or extremely terse text.

We used an ***80/20*** approach. We assume that if you're going for a PhD in Java, this won't be your only book. So we don't talk about *everything*. Just the stuff you'll actually *use*.



Here's what YOU can do to bend your brain into submission.

So, we did our part. The rest is up to you. These tips are a starting point; Listen to your brain and figure out what works for you and what doesn't. Try new things.

cut this out and stick it on your refridgerator.

1. **Slow down. The more you understand, the less you have to memorize.**
   Don't just *read*. Stop and think. When the book asks you a question, don't just skip to the answer. Imagine that someone really *is* asking the question. The more deeply you force your brain to think, the better chance you have of learning and remembering.

2. **Do the exercises. Write your own notes.**
   We put them in, but if we did them for you, that would be like having someone else do your workouts for you. And don't just *look* at the exercises. **Use a pencil.** There's plenty of evidence that physical activity *while* learning can increase the learning.

3. **Read the "There are No Dumb Questions"**
   That means all of them. They're not optional side-bars—they're part of the core content! Sometimes the questions are more useful than the answers.

4. **Don't do all your reading in one place.**
   Stand-up, stretch, move around, change chairs, change rooms. It'll help your brain *feel* something, and keeps your learning from being too connected to a particular place.

5. **Make this the last thing you read before bed. Or at least the last *challenging* thing.**
   Part of the learning (especially the transfer to long-term memory) happens *after* you put the book down. Your brain needs time on its own, to do more processing. If you put in something new during that processing-time, some of what you just learned will be lost.

6. **Drink water. Lots of it.**
   Your brain works best in a nice bath of fluid. Dehydration (which can happen before you ever feel thirsty) decreases cognitive function.

7. **Talk about it. Out loud.**
   Speaking activates a different part of the brain. If you're trying to understand something, or increase your chance of remembering it later, say it out loud. Better still, try to explain it out loud to someone else. You'll learn more quickly, and you might uncover ideas you hadn't known were there when you were reading about it.

8. **Listen to your brain.**
   Pay attention to whether your brain is getting overloaded. If you find yourself starting to skim the surface or forget what you just read, it's time for a break. Once you go past a certain point, you won't learn faster by trying to shove more in, and you might even hurt the process.

9. **Feel something!**
   Your brain needs to know that this *matters*. Get involved with the stories. Make up your own captions for the photos. Groaning over a bad joke is *still* better than feeling nothing at all.

10. **Type and run the code.**
    Type and run the code examples. Then you can experiment with changing and improving the code (or breaking it, which is sometimes the best way to figure out what's really happening). For long examples or Ready-bake code, you can download the source files from headfirstjava.com

What you need for this book:

You do *not* need any other development tool, such as an Integrated Development Environment (IDE). We strongly recommend that you *not* use anything but a basic text editor until you complete this book (and *especially* not until after chapter 16). An IDE can protect you from some of the details that really matter, so you're much better off learning from the command-line and then, once you really understand what's happening, move to a tool that automates some of the process.

SETTING UP JAVA

- If you don't already have a **1.5** or greater **Java 2 Standard Edition SDK** (Software Development Kit), you need it. If you're on Linux, Windows, or Solaris, you can get it for free from java.sun.com (Sun's website for Java developers). It usually takes no more than two clicks from the main page to get to the J2SE downloads page. Get the latest *non-beta* version posted. The SDK includes everything you need to compile and run Java.
  If you're running Mac OS X 10.4, the Java SDK is already installed. It's part of OS X, and you don't have to do *anything* else. If you're on an earlier version of OS X, you have an earlier version of Java that will work for 95% of the code in this book.
  Note: This book is based on Java 1.5, but for stunningly unclear marketing reasons, shortly before release, Sun renamed it Java 5, while still keeping "1.5" as the version number for the developer's kit. So, if you see Java 1.5 or Java 5 or Java 5.0, or "Tiger" (version 5's original code-name), *they all mean the same thing.* There was never a Java 3.0 or 4.0—it jumped from version 1.4 to 5.0, but you will still find places where it's called 1.5 instead of 5. Don't ask.
  (Oh, and just to make it more entertaining, Java 5 and the Mac OS X 10.4 were both given the same code-name of "Tiger", and since OS X 10.4 is the version of the Mac OS you need to run Java 5, you'll hear people talk about "Tiger on Tiger". It just means Java 5 on OS X 10.4).

- The SDK does *not* include the **API documentation,** and you need that! Go back to java.sun.com and get the J2SE API documentation. You can also access the API docs online, without downloading them, but that's a pain. Trust us, it's worth the download.

- You need a **text editor**. Virtually any text editor will do (vi, emacs, pico), including the GUI ones that come with most operating systems. Notepad, Wordpad, TextEdit, etc. all work, as long as you make sure they don't append a ".txt" on to the end of your source code.

- Once you've downloaded and unpacked/zipped/whatever (depends on which version and for which OS), you need to add an entry to your **PATH** environment variable that points to the /bin directory inside the main Java directory. For example, if the J2SDK puts a directory on your drive called "j2sdk1.5.0", look inside that directory and you'll find the "bin" directory where the Java binaries (the tools) live. Tha bin directory is the one you need a PATH to, so that when you type:

```
% javac
```

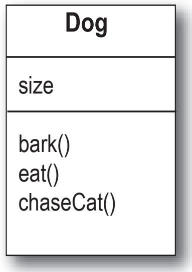
  at the command-line, your terminal will know how to find the *javac* compiler.
  Note: if you have trouble with you installation, we recommend you go to javaranch.com, and join the Java-Beginning forum! Actually, you should do that whether you have trouble or not.

Note: much of the code from this book is available at wickedlysmart.com

Last-minute things you need to know:

This is a learning experience, not a reference book. We deliberately stripped out everything that might get in the way of *learning* whatever it is we're working on at that point in the book. And the first time through, you need to begin at the beginning, because the book makes assumptions about what you've already seen and learned.

We use a simpler, modified faux-UML



We use simple UML-*like* diagrams.

If we'd used *pure* UML, you'd be seeing something that *looks* like Java, but with syntax that's just plain *wrong*. So we use a simplified version of UML that doesn't conflict with Java syntax. If you don't already know UML, you won't have to worry about learning Java *and* UML at the same time.

We don't worry about organizing and packaging your own code until the end of the book.

In this book, you can get on with the business of learning Java, without stressing over some of the organizational or administrative details of developing Java programs. You *will*, in the real world, need to know—and use—these details, so we cover them in depth. But we save them for the end of the book (chapter 17). Relax while you ease into Java, gently.

You should do ALL of the "Sharpen your pencil" activities

Sharpen your pencil

The end-of-chapter exercises are mandatory; puzzles are optional. Answers for both are at the end of each chapter.

One thing you need to know about the puzzles—*they're puzzles.* As in logic puzzles, brain teasers, crossword puzzles, etc. The *exercises* are here to help you practice what you've learned, and you should do them all. The puzzles are a different story, and some of them are quite challenging in a *puzzle* way. These puzzles are meant for *puzzlers*, and you probably already know if you are one. If you're not sure, we suggest you give some of them a try, but whatever happens, don't be discouraged if you *can't* solve a puzzle or if you simply can't be bothered to take the time to work them out.

Activities marked with the Exercise (running shoe) logo are mandatory! Don't skip them if you're serious about learning Java.

Exercise

The 'Sharpen Your Pencil' exercises don't have answers.

Not printed in the book, anyway. For some of them, there *is* no right answer, and for the others, part of the learning experience for the Sharpen activities is for *you* to decide if and when your answers are right. (Some of our *suggested* answers are available on wickedlysmart.com)

If you see the Puzzle logo, the activity is optional, and if you don't like twisty logic or cross-word puzzles, you won't like these either.

The code examples are as lean as possible

It's frustrating to wade through 200 lines of code looking for the two lines you need to understand. Most examples in this book are shown within the smallest possible context, so that the part you're trying to learn is clear and simple. So don't expect the code to be robust, or even complete. That's *your* assignment for after you finish the book. The book examples are written specifically for *learning*, and aren't always fully-functional.

Technical Editors

"Credit goes to all, but mistakes are the sole reponsibility of the author...". Does anyone really believe that? See the two people on this page? If you find technical problems, it's probably *their* fault. :)

Jess works at Hewlett-Packard on the Self-Healing Services Team. She has a Bachelor's in Computer Engineering from Villanova University, has her SCPJ 1.4 and SCWCD certifications, and is literally months away from receiving her Masters in Software Engineering at Drexel University (whew!)

When she's not working, studying or motoring in her MINI Cooper S, Jess can be found fighting her cat for yarn as she completes her latest knitting or crochet project (anybody want a hat?) She is originally from Salt Lake City, Utah (no, she's not Mormon... yes, you were too going to ask) and is currently living near Philadelphia with her husband, Mendra, and two cats: Chai and Sake.

You can catch her moderating technical forums at javaranch.com.

Valentin Valentin Crettaz has a Masters degree in Information and Computer Science from the Swiss Federal Institute of Technology in Lausanne (EPFL). He has worked as a software engineer with SRI International (Menlo Park, CA) and as a principal engineer in the Software Engineering Laboratory of EPFL.

Valentin is the co-founder and CTO of Condris Technologies, a company specializing in the development of software architecture solutions.

His research and development interests include aspect-oriented technologies, design and architectural patterns, web services, and software architecture. Besides taking care of his wife, gardening, reading, and doing some sport, Valentin moderates the SCBCD and SCDJWS forums at Javaranch.com. He holds the SCJP, SCJD, SCBCD, SCWCD, and SCDJWS certifications. He has also had the opportunity to serve as a co-author for Whizlabs SCBCD Exam Simulator.

(We're still in shock from seeing him in a *tie*.)

Other people to ~~blame~~ credit:

At O'Reilly:

Our biggest thanks to **Mike Loukides** at O'Reilly, for taking a chance on this, and helping to shape the Head First concept into a book (and *series*). As this second edition goes to print there are now five Head First books, and he's been with us all the way. To **Tim O'Reilly**, for his willingness to launch into something *completely* new and different. Thanks to the clever **Kyle Hart** for figuring out how Head First fits into the world, and for launching the series. Finally, to **Edie Freedman** for designing the Head First "emphasize the *head*" cover.

Our intrepid beta testers and reviewer team:

Our top honors and thanks go to the director of our javaranch tech review team, **Johannes de Jong.** This is your fifth time around with us on a Head First book, and we're thrilled you're still speaking to us. **Jeff Cumps** is on his third book with us now and relentless about finding areas where we needed to be more clear or correct.

Corey McGlone, you rock. And we think you give the clearest explanations on javaranch. You'll probably notice we stole one or two of them. **Jason Menard** saved our technical butts on more than a few details, and **Thomas Paul**, as always, gave us expert feedback and found the subtle Java issues the rest of us missed. **Jane Griscti** has her Java chops (and knows a thing or two about *writing*) and it was great to have her helping on the new edition along with long-time javarancher **Barry Gaunt**.

Marilyn de Queiroz gave us excellent help on *both* editions of the book. **Chris Jones**, **John Nyquist, James Cubeta, Terri Cubeta,** and **Ira Becker** gave us a ton of help on the first edition.

Special thanks to a few of the Head Firsters who've been helping us from the beginning: **Angelo Celeste**, **Mikalai Zaikin,** and **Thomas Duff** (twduff.com). And thanks to our terrific agent, David Rogelberg of StudioB (but seriously, what about the *movie* rights?)

Some of our Java expert reviewers...

Just when you thought there wouldn't be any more acknowledgements*.

More Java technical experts who helped out on the first edition (in pseudo-random order):

Emiko Hori, Michael Taupitz, Mike Gallihugh, Manish Hatwalne, James Chegwidden, Shweta Mathur, Mohamed Mazahim, John Paverd, Joseph Bih, Skulrat Patanavanich, Sunil Palicha, Suddhasatwa Ghosh, Ramki Srinivasan, Alfred Raouf, Angelo Celeste, Mikalai Zaikin, John Zoetebier, Jim Pleger, Barry Gaunt, and Mark Dielen.

The first edition puzzle team:

Dirk Schreckmann, Mary "JavaCross Champion" Leners, Rodney J. Woodruff, Gavin Bong, and Jason Menard. Javaranch is lucky to have you all helping out.

Other co-conspirators to thank:

Paul Wheaton, the javaranch Trail Boss for supporting thousands of Java learners.
Solveig Haugland, mistress of J2EE and author of "Dating Design Patterns".
Authors **Dori Smith** and **Tom Negrino (backupbrain.com)**, for helping us navigate the tech book world.
Our Head First partners in crime, **Eric Freeman and Beth Freeman** (authors of Head First Design Patterns), for giving us the Bawls™ to finish this on time.
Sherry Dorris, for the things that really matter.

Brave Early Adopters of the Head First series:

Joe Litton, Ross P. Goldberg, Dominic Da Silva, honestpuck, Danny Bromberg, Stephen Lepp, Elton Hughes, Eric Christensen, Vulinh Nguyen, Mark Rau, Abdulhaf, Nathan Oliphant, Michael Bradly, Alex Darrow, Michael Fischer, Sarah Nottingham, Tim Allen, Bob Thomas, and Mike Bibby (the first).

*The large number of acknowledgements is because we're testing the theory that everyone mentioned in a book acknowledgement will buy at least one copy, probably more, what with relatives and everything. If you'd like to be in the acknowledgement of our *next* book, and you have a large family, write to us.