Java 5.0 Tiger

A Developer's Notebook™

Brett McLaughlin
David Flanagan

O'REILLY®

Java

Java 5.0 Tiger *A Developer's Notebook*

The next version of Java, code-named Tiger, is more than just a minor update. With over 100 substantial changes to the core language, as well as numerous library and API additions, developers have a variety of new features, facilities, and techniques available. But with so many changes, where do you start? You could read through the lengthy and often boring language specification; you could wait for the latest 500-page tome on concepts and theory; you could even play around with the new JDK, hoping you figure things out—or you can get straight to work with *Java 5.0 Tiger: A Developer's Notebook*.

This no-nonsense, down-and-dirty guide by bestselling Java authors Brett McLaughlin and David Flanagan skips all the boring prose and lecture and jumps right into Tiger. You'll have a handle on many of the important new features of the language by the end of the first chapter, and be neck-deep in code before you hit page 20. Through more than 50 working code samples, you'll get complete practical coverage of generics, learn how boxing and unboxing affects your type conversions, understand the power of varargs, learn how to write enumerated types and annotations, master Java's new formatting methods and the `for/in` loop, and even get a grip on concurrency in the JVM.

Before you're through, you'll understand:

- Generics, including type-safe collections and defining your own generic classes
- Enumerated types and values, and their relationship to `public static final` constants
- Tiger's autoboxing and autounboxing conversions between primitive types and wrapper types
- Variable arguments (varargs), including defining your own vararg methods
- Tiger's extensive support for compiler-checked annotations
- The `for/in` loop
- The new `format()` and `printf()` methods
- The extensive concurrency support in Tiger, including locks, scheduling timers, uncaught exceptions in threads, and the new `Callable` interface

A Developer's Notebook is just what it claims to be—the often frantic scribbling and notes that a true-blue alpha geek knows you'll need when working with a new language, API, or project—available BEFORE you begin. It's the code that solves problems, stripped of commentary that can serve as more of a paperweight than an epiphany. It's hackery, focused not on what is nifty or might be fun if you've got some spare time, but what you need to simply "make it work."

US $29.99 **CAN $37.99**

ISBN: 978-0-596-00738-6

Visit O'Reilly on the Web at www.oreilly.com

O'REILLY®

Java 5.0 Tiger

A Developer's Notebook™

Java 5.0 Tiger

A Developer's Notebook™

Brett McLaughlin and David Flanagan

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Sebastopol · Taipei · Tokyo

Java 5.0 Tiger: A Developer's Notebook™
by Brett McLaughlin and David Flanagan

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Editor: Brett McLaughlin

Production Editor: Reg Aubry

Cover Designer: Edie Freedman

Interior Designer: Melanie Wang

Printing History:

June 2004: First Edition.

ISBN: 978-0-596-00738-6
[M] [2/08]

Contents

The Developer's Notebook Series

So, you've managed to pick this book up. Cool. Really, I'm excited about that! Of course, you may be wondering why these books have the odd-looking, college notebook sort of cover. I mean, this is O'Reilly, right? Where are the animals? And, really, do you *need* another series? Couldn't this just be a cookbook? How about a nutshell, or one of those cool hacks books that seems to be everywhere? The short answer is that a developer's notebook is none of those things—in fact, it's such an important idea that we came up with an entirely new look and feel, complete with cover, fonts, and even some notes in the margin. This is all a result of trying to get something into your hands you can actually use.

It's my strong belief that while the nineties were characterized by everyone wanting to learn everything (Why not? We all had six-figure incomes from dot-com companies), the new millennium is about information pain. People don't have time (or the income) to read through 600 page books, often learning 200 things, of which only about 4 apply to their current job. It would be much nicer to just sit near one of the uber-coders and look over his shoulder, wouldn't it? To ask the guys that are neck-deep in this stuff why they chose a particular method, how they performed this one tricky task, or how they avoided that threading issue when working with piped streams. The thinking has always been that books can't serve that particular need—they can inform, and let you decide, but ultimately a coder's mind was something that couldn't really be captured on a piece of paper.

This series says that assumption is patently wrong—and we aim to prove it.

A Developer's Notebook is just what it claims to be: the often-frantic scribbling and notes that a true-blue alpha geek mentally makes when working with a new language, API, or project. It's the no-nonsense code that solves problems, stripped of page-filling commentary that often serves more as a paperweight than an epiphany. It's hackery, focused not on what is nifty or might be fun to do when you've got some free time (when's the last time that happened?), but on what you need to simply "make it work." This isn't a lecture, folks—it's a lab. If you want a lot of concept, architecture, and UML diagrams, I'll happily and proudly point you to our animal and nutshell books. If you want every answer to every problem under the sun, our omnibus cookbooks are killer. And if you are into arcane and often quirky uses of technology, hacks books simply rock. But if you're a coder, down to your core, and you just want to get on with it, then you want a Developer's Notebook. Coffee stains and all, this is from the mind of a developer to yours, barely even cleaned up enough for print. I hope you enjoy it...we sure had a good time writing them.

Notebooks Are...

Example-driven guides

As you'll see in the "Organization" section, developer's notebooks are built entirely around example code. You'll see code on nearly every page, and it's code that *does something*—not trivial "Hello World!" programs that aren't worth more than the paper they're printed on.

Aimed at developers

Ever read a book that seems to be aimed at pointy-haired bosses, filled with buzzwords, and feels more like a marketing manifesto than a programming text? We have too—and these books are the antithesis of that. In fact, a good notebook is incomprehensible to someone who can't program (don't say we didn't warn you!), and that's just the way it's supposed to be. But for developers...it's as good as it gets.

Actually enjoyable to work through

Do you really have time to sit around reading something that isn't any fun? If you do, then maybe you're into thousand-page language references—but if you're like the rest of us, notebooks are a much better fit. Practical code samples, terse dialogue centered around practical examples, and even some humor here and there—these are the ingredients of a good developer's notebook.

About doing, not talking about doing

If you want to read a book late at night without a computer nearby, these books might not be that useful. The intent is that you're coding as you go along, knee deep in bytecode. For that reason, notebooks talk code, code, code. Fire up your editor before digging in.

Notebooks Aren't...

Lectures

We don't let just anyone write a developer's notebook—you've got to be a bona fide programmer, and preferably one who stays up a little too late coding. While full-time writers, academics, and theorists are great in some areas, these books are about programming in the trenches, and are filled with instruction, not lecture.

Filled with conceptual drawings and class hierarchies

This isn't a nutshell (there, we said it). You won't find 100-page indices with every method listed, and you won't see full-page UML diagrams with methods, inheritance trees, and flow charts. What you will find is page after page of source code. Are you starting to sense a recurring theme?

Long on explanation, light on application

It seems that many programming books these days have three, four, or more chapters before you even see any working code. I'm not sure who has authors convinced that it's good to keep a reader waiting this long, but it's not anybody working on *this* series. We believe that if you're not coding within ten pages, something's wrong. These books are also chock-full of practical application, taking you from an example in a book to putting things to work on your job, as quickly as possible.

Organization

Developer's Notebooks try to communicate different information than most books, and as a result, are organized differently. They do indeed have chapters, but that's about as far as the similarity between a notebook and a traditional programming book goes. First, you'll find that all the headings in each chapter are organized around a specific task. You'll note that we said *task*, not *concept*. That's one of the important things to get about these books—they are first and foremost about doing something. Each of these headings represents a single *lab*. A lab is just what it sounds like—steps to accomplish a specific goal. In fact, that's the first

heading you'll see under each lab: "How do I do that?" This is the central question of each lab, and you'll find lots of down-and-dirty code and detail in these sections.

Some labs have some things not to do (ever played around with potassium in high school chemistry?), helping you avoid common pitfalls. Some labs give you a good reason for caring about the topic in the first place; we call this the "Why do I care?" section, for obvious reasons. For those times when code samples don't clearly communicate what's going on, you'll find a "What just happened" section. It's in these sections that you'll find concepts and theory—but even then, they are tightly focused on the task at hand, not explanation for the sake of page count. Finally, many labs offer alternatives, and address common questions about different approaches to similar problems. These are the "What about..." sections, which will help give each task some context within the programming big picture.

And one last thing—on many pages, you'll find notes scrawled in the margins of the page. These aren't for decoration; they contain tips, tricks, insights from the developers of a product, and sometimes even a little humor, just to keep you going. These notes represent part of the overall communication flow—getting you as close to reading the mind of the developer-author as we can. Hopefully they'll get you that much closer to feeling like you are indeed learning from a master.

And most of all, remember—these books are...

All Lab, No Lecture

—Brett McLaughlin, Series Creator

Preface

Professional Java
Enterprise Java
Commercial Java

These are all terms that are commonplace in programming discussions these days—and for good reason. Gone are the days when Java was considered a toy language for creating web games, futilely trying to catch up to its "big brothers," C and C++. While AWT and Swing (and now SWT) are important parts of the Java language, Java has also evolved to take on more far-ranging tasks—database interaction, financial management, e-commerce, and more. Its speed is comparable to C, and its APIs are far-reaching. As a result, the core language has undergone significant stabilization, and Java 1.3, and then 1.4, were largely steps towards maturing the platform, rather than radically changing it.

Enter Java 5.0—code-named Tiger. Actually, it's Java 5, version 1.5. Well, it's the J2SE, which I suppose makes it Java 2, Standard Edition, 5, version 1.5. Confusing enough for you? Thankfully, whatever the thing is called, the additions are worthy of all the hubbub; this isn't your father's Java (or to be more accurate, it's not your slightly older brother's Java) anymore.

Looking more like a completely new product than just a revision of an older language, Tiger is chock-full of dramatic changes to what you know as simply Java. You can't just read through the release notes and figure this one out; and since the new features are a lot more important than all the oddities about its versioning, I'll just call it Tiger throughout the book, and sidestep Java 2 version 5...er...version 1.5...well...as I said, Tiger.

Whatever Tiger ends up being called officially, it introduces so many new features to the language that it took nearly 200 pages to cover them—and you'll find that each page of this book is dense with code, example, and terse explanation. There isn't any wasted space. In fact, that's precisely what you're holding in your hands—a concise crash course in the next evolution of Java, Tiger. By the time you're through, you'll be typing your lists, taking your overloading to an entirely new level, writing compile-time checked annotations, and threading more efficiently than ever. And that doesn't take into account how much fun it is to type all sorts of new characters into your source code. You haven't lived until `@`, `<`, `>`, and `%` are strewn throughout your editor...well, maybe that's just me wanting to have a little more fun at the workplace. Whatever your reason for getting into Tiger, though, you'll find more tools at your disposal than ever before, and far more change in any version of Java since its initial 1.0 release. Fire up your code editor, buckle your seat belts, and get ready to hit the ground running.

Let's tame the Tiger.

Organization

This book is set up to be something of a cross between a learning exercise (where you would read from front to back), and a cookbook (where you can skip around without concern). For the most part, you can feel free to look through the table of contents or index, and find what you're looking for. However, as many of the subjects in this book are interrelated (such as generics, the `for/in` statement, and autoboxing), you may find yourself reading an article that assumes knowledge from a previous section of the book. In these cases, just browse the referenced chapter, and you should be all set. A little extra learning is a good thing anyway, right?

How This Book Was Written

This book is the result of an unusual, but fruitful collaboration between David Flanagan and Brett McLaughlin. David was at work on the fifth edition of Java in a Nutshell, but was eager to get coverage of the major language changes in Tiger out sooner than the production schedule for that book allowed. Brett, meanwhile, was the driving editorial force behind this innovative new series of Developer's Notebooks, and was eager to include a title on Tiger in the series.

The process went like this:

- David researched the new features of Tiger and wrote about them for Java in a Nutshell. He sent drafts of his new material to Brett.
- Brett feverishly ripped those chapters apart, rewrote almost everything, added new examples, and reassembled everything into the Developer's Notebook format.

The result is a book almost entirely written by Brett, based on research done by David. The tone of the writing and the engaging style of the book is Brett's, and readers of this book and Java in a Nutshell will be hard-pressed to find any duplication of prose. In a few cases, Brett has used code samples that also appear in Java in a Nutshell, and in each case that fact is mentioned in the margin.

About the Examples

This book has hundreds of code examples, spread throughout its pages. While some complete code listings are shown in the text, other examples are shown only in part. While some readers may enjoy typing in these programs on their own, many of us just don't have the time. Because of this, every single example, and almost all of the partial examples, are ready for compilation in Java source files, ready for download.

Additionally, the process of compilation (especially class path issues) remains one of Java's most problematic features. To help you out, an Ant buildfile is included with the samples, called *build.xml*. You'll need to download and install Ant (available at *http://ant.apache.org*) to take advantage of this buildfile, and I strongly urge you to do just that. Ant installation is easy, and you can always refer to *Ant: The Definitive Guide* (O'Reilly) if you need assistance. Your directory structure should look something like this:

```
<basedir>
  |
  +--src (contains build.xml)
  |
  +--classes
```

TIP

This is all taken care of for you if you just download the code and unzip it.

Navigate to your local *src* directory, and type `ant`. You'll get an error if you don't have Ant set up properly. Otherwise, you should see something like the following:

```
${basedir}\code\src>ant
Buildfile: build.xml

compile:
     [echo] Compiling all Java files...
    [javac] Compiling 41 source files to code\classes
    [javac] Note: code\src\com\oreilly\tiger\ch06\DeprecatedTester.java
                uses or overrides a deprecated API.
    [javac] Note: Recompile with -Xlint:deprecation for details.
    [javac] Note: Some input files use unchecked or unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.

BUILD SUCCESSFUL
Total time: 9 seconds
```

I'll leave it to you to explore the other targets within *build.xml*; there are also notes in most chapters about targets that apply to that chapter, or to a specific example. All this code is heavily tested, and mildly documented. Just make sure you've got Tiger as the first Java compiler on your classpath, or you'll get all sorts of nasty errors!

You may download this sample code, as well as check out errata, view related resources and online articles, and see the latest on this book, at *http://www.oreilly.com/catalog/javaadn/*. Check this site often, as lots of new content may be available as time goes by and we update the examples.

Conventions Used in This Book

Italic is used for:

- Pathnames, filenames, program names, compilers, options, and commands
- New terms where they are defined
- Internet addresses, such as domain names and URLs

Boldface is used for:

- Particular keys on a computer keyboard
- Names of user interface buttons and menus

`Constant width` is used for:

- Anything that appears literally in a JSP page or a Java program, including keywords, data types, constants, method names, variables, class names, and interface names
- Command lines and options that should be typed verbatim on the screen
- All JSP and Java code listings
- HTML documents, tags, and attributes

`Constant width italic` is used for:

- General placeholders that indicate that an item is replaced by some actual value in your own program

`Constant width bold` is used for:

- Text that is typed in code examples by the user

TIP

This icon designates a note, which is an important aside to the nearby text.

WARNING

This icon designates a warning relating to the nearby text.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)

We have a web page for this book, where we list errata, examples, or any additional information. You can access this page at:

*http://www.oreilly.com/catalog/*javaadn/

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

http://www.oreilly.com/

Acknowledgments from Brett

The "I" you see in these pages is me—for better or for worse, I came up with this series, and am thrilled to be able to bring one of the first books in the series to you. But, that leads me to the enormously talented group of folks who made that possible.

There was a time when I loved writing acknowledgements, because I got to thank everybody involved in helping me get through another book. Of course, now I realize that there are so many people I forget to thank, that I'm a little scared...I guess that's the Oscar-acceptance-paranoia working itself out. In any case, any book such as this truly is a tremendous effort by a ton of people, and I couldn't go without at least *trying* to name most of them.

To Mike Loukides, who edits most of my books (this being the exception), and Mike Hendrickson, who's just all-around smart—thanks for paving the way for these new, inventive, cool little notebooks. I think you've done the programming world a real service with them. I need to thank David Flanagan for doing all the heavy lifting; the Sun folks, especially at CAP, for letting me see JDK 1.5 early on; and guys like Hans Bergsten, Bruce Perry, Bob McWhirter, and Steve Holzner for writing good books and letting me spend less time editing than I deserve to.

Finally, in trying to keep things brief (you'll think I'm funny because of that, right?), I owe the biggest debt to my family, as is always the case. My wife, Leigh, only gripes occasionally when I'm working at 9:00 at night. Of course, that's mostly because she's exhausted from chasing the two bits of inspiration I have; my older son, Dean, and my younger son, Robbie. When you guys can read, you'll see your names here, so thank the readers for the college fund, OK?

Acknowledgments from David

Thanks first and foremost to Brett for his enthusiasm, and for working overtime and pulling this book together so quickly. Thanks also to Mike Loukides for supporting the endeavor, and to Deb Cameron, my editor for *Java in a Nutshell*, for allowing me the time to work on it.