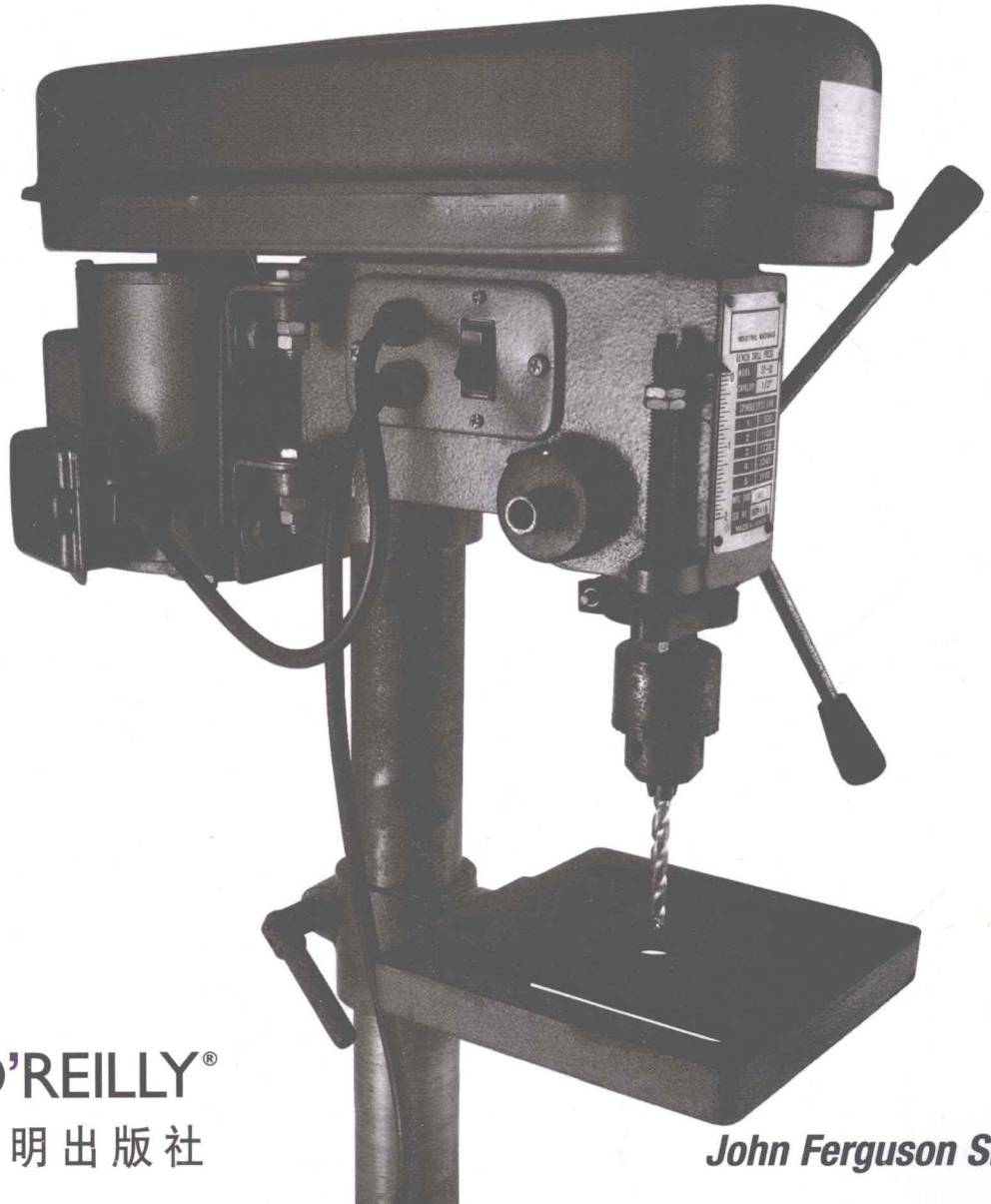
上卷

JAVA™ POWER TOOLS

Java超级工具（影印版）



O'REILLY®
开明出版社

John Ferguson Smart 著

上卷

Java 超级工具（影印版）

Java™ Power Tools

John Ferguson Smart

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc.授权北京凤凰天下文化发展有限公司、开明出版社出版发行

开明出版社

图书在版编目（CIP）数据

Java超级工具=Java Power Tools：英文 /（美）斯马特（Smart，J.F.）著.—影印本.—北京：开明出版社，2009.3

ISBN 978-7-80205-731-9

Ⅰ. J… Ⅱ.斯… Ⅲ.JAVA语言—程序设计—英文 Ⅳ.TP312

中国版本图书馆CIP数据核字（2009）第035520号

江苏省版权局著作权合同登记

图字：10-2009-084 号

书名：Java 超级工具（影印版）

出版：开明出版社出版（北京海淀区西三环北路 19 号　邮编 100089）

经销：全国新华书店

印刷：北京市梦宇印务有限公司（北京市通州区张家湾镇张辛庄村）

开本：787×1092　1 / 16

印张：58

字数：974 千字

版次：2009 年 4 月　北京第 1 版

印次：2009 年 4 月　北京第 1 次印刷

定价：98.00 元（上下册）

O'Reilly Media, Inc.介绍

O'Reilly Media, Inc.是世界上在 UNIX、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时是联机出版的先锋。

从最畅销的《The Whole Internet User's Guide & Catalog》（被纽约公共图书馆评为二十世纪最重要的50本书之一）到GNN（最早的Internet门户和商业网站），再到WebSite（第一个桌面PC的Web服务器软件），O'Reilly Media, Inc.一直处于Internet发展的最前沿。

许多书店的反馈表明，O'Reilly Media, Inc.是最稳定的计算机图书出版商 —— 每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly Media, Inc.具有深厚的计算机专业背景，这使得O'Reilly Media, Inc.形成了一个非常不同于其他出版商的出版方针。O'Reilly Media, Inc.所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly Media, Inc.还有许多固定的作者群体 —— 他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly Media, Inc.依靠他们及时地推出图书。因为O'Reilly Media, Inc.紧密地与计算机业界联系着，所以O'Reilly Media, Inc.知道市场上真正需要什么图书。

This book is dedicated to my wonderful wife Chantal, and my two lovely boys, James and William, who are my constant source of inspiration, wonder, and joy.

Foreword

Designing, coding, and deploying working Java applications isn't easy. Doing it in a predictable manner rapidly with an acceptable level of quality is even harder. In addition to having to understand what stakeholders want or the varying skills of team members or even the myriad web, data access, and utility frameworks one can choose from, you've got to actually manage the development process itself!

The coding of requirements is challenging enough, but as anyone who's ever delivered a working application knows, in the grand scheme of things, that's one sliver of the development process—in fact, some may say that's the easiest part. Think about all the techniques and processes that aggregate up to produce a software application.

First, you've got to figure out how to deliver the working application in a predictable manner. At a high level, this means three things: tracking changes to source code assets, keeping up with any uncovered issues, defects, or feature requests, and assembling the application in a reliable and repeatable manner.

Next, you're going to want to actually ensure the application under development actually works—ideally *during* development. This means writing tests early. Of course, this process is easier said than done. Although arguably there are few standard testing frameworks from which to chose, there is a cornucopia of associated tools that accelerate writing developer tests by addressing specific challenges.

What's more, as the code base grows, you'll probably want to understand what's being coded and how well it's being developed. Although tests cancertainly verify code functionality, you may also want lighter-weight tools that can report on various metrics, such as complexity, coding standards, or even the coverage of tests.

Of course, if you've got a mechanism for assembling your application in a repeatable and reliable manner, it makes sense to augment this process by running tests and even analysis tools. What's more, given that you want to produce working code quickly, it makes sense to assemble the application *often*—in fact, assembling it continuously facilitates discovering issues as they arise.

Finally, you're going to want to enable easy maintenance of the code base so that features can be added often—in the same rapid and repeatable manner that the application was built.

John has assembled what I think is the "A" list of tools and techniques that will help you meet each and everyone of the challenges above. In fact, John presents multiple choices in some cases, giving you the opportunity to decide which tool works best for you. Whether you decide to use Ant or Maven for delivering a working application in a predictable manner, TestNG or JUnit for early developer testing, PMD or FindBugs for code analysis, or CruiseControl or Luntbuild for Continuous Integration, this book addresses the fundamental techniques for effectively employing these tools (and a multitude of others as well).

Rapid development of Java applications (which have an acceptable level of associated quality) is still hard as ever; however, after reading John's magnum opus on the tools and techniques that ultimately enable predictability, confident, and accelerated delivery in a software development process, you'll find that designing, coding, and deploying high-quality Java applications rapidly just got a whole lot easier.

—Andrew Glover, *President, Stelligent Incorporated*

Preface

> Here is Edward Bear* coming downstairs now, bump, bump, bump, on the back of his head, behind Christopher Robin. It is, as far as he knows, the only way of coming downstairs, but sometimes he feels that there really is another way, if only he could stop bumping for a moment and think of it.
>
> —"We are introduced to Winnie-the-Pooh and some bees, and the stories begin,"
> *Winnie the Pooh*, A. A. Milne

Thus does A. A. Milne introduce that classic character of children's literature, Winnie the Pooh. As you can see, Winnie the Pooh seems to have some issues with the way he goes downstairs (we probably wouldn't be too far off if we were to speak of "pain points").

Software development sometimes feels like this. It is easy to get so bogged down in the details, under the pressure of tight deadlines and changing requirements, that you forget that there might just be a better way. A high number of bugs, a difficult integration process, long and painful builds, and poor project visibility become an accepted part of the developer's life.

The good news is that there are in fact many easy ways to improve your software development lifecycle.

A judicious use of tools can go a long way in boosting developer productivity. For example, many distributed development teams use nothing more sophisticated than a CVS or Subversion repository for storing application code and a mailing list for discussion. Imagine a system in which, whenever someone commits a change, issues are automatically closed or updated based on the commit comment. The issue management system then automatically notifies the issue owner of the change of status. Meanwhile, a dedicated build server detects the commit, builds and tests the system, and notifies the developer of any failures. In a few relatively simple steps, you've taken your development platform to a new level of reactivity and responsiveness.

* For nonnative English readers: Edward Bear is a round-about way of saying "Teddy Bear." In the original book, this text is accompanied by a drawing of Christopher Robin, a boy of about four years old, going down a flight of stairs dragging his teddy bear behind him.

This is just a first step. Soon you will be integrating automatically running unit and possibly integration tests, code coverage tools, style checking tools, and more to create a highly reactive, informative, finely tuned project infrastructure.

Tools are the key to making such an infrastructure work efficiently. Once you start to adopt a set of SDLC tools that suit your project and your organization, you will see a revolutionary shift in a groups development practices. Agile development authors such as Alistair Cockburn rightly point to optimal communication as being the cornerstone of productive development. Developing in a team without continuous integration is like rock climbing without a rope. In addition, developing in a team without collaboration and connective development infrastructure is like developing in a boring beige office space where everyone comes to work on time, enters a closed office, closes the door, and starts programming without ever stopping to have meetings. In other words, programming without a good set of SDLC tools is very much the equivalent of fighting a modern war with swords and armor—the adoption of SDLC tools is a generational shift, the programmers just coming into the workforce will never know an alternative, but the programmers who haven't experienced this shift are in danger of missing the trend altogether.

Now you shouldn't go thinking that SDLC tools are only for large teams or big organizations. They aren't. In fact, most SDLC tools are easy to setup, and almost all organizations can benefit, even a small outfit with only two or three developers.

Most of these tools and techniques are not particularly difficult to put into place, but they do require a minimum of effort, to step back and take a long, hard look at your current practices. Chances are, there are things that can be improved.

This book is part of the O'Reilly Power Tools series, which began with the illustrious Unix Power Tools back in 1993. It has no relation to the "Java Power Tools" library (*http://www.ccs.neu.edu/jpt/*), which is a software research project in the Java GUI field, conducted by College of Computer & Information Science at the Northeastern University, Boston, under the direction of Richard Rasala.

How This Book Is Organized

One of the most characteristic traits of the open source (*http://opensource.org*) Java world is choice. It seems that, for any given task, there are always at least two open source tools that can do the job. To reflect this, I have tried to give a representative survey of the available open source tools for each area of software development that we cover. The book does not try to "sell" any one tool, or even any one tool in each domain. On the contrary, we try to give readers enough information so that they can decide for themselves which tool is the most appropriate for their particular organization, and give enough practical information to get them up and running.

This book is organized into sections, with each section covering a particular aspect of the software development lifecycle (or SDLC). With each section, we look at a number of available tools that can be used to improve this aspect of the SDLC.

Build Tools

In the first section, we cover possibly the most fundamental tool of all (after the compiler and the IDE, that is): the build tool. Indeed, when used well, the build tool becomes the cornerstone of your SDLC process. It is this tool that coordinates, federates, and binds the other SDLC tools together into a single, coherent process. And the build tool helps to ensure that your project can be built on any machine, in any environment.

In the Java world, two tools dominate this area. The first is Ant, the traditional Java build tool, which uses a straightforward procedural approach and benefits from a very large user base and a rich set of extensions. The second is Maven 2, which uses a powerful, declarative approach to project build management and, as we will see, goes much further than being a simple build tool. We will look at each of them in some detail in this section.

Version Control Tools

The second section covers that other fundamental component of any software development infrastructure: a version control system not only provides critical backups of your source code, it also lets developers work together on the same project without getting in each other's way. Version control systems also allow you to identify versions and coordinate releases and (if necessary) rollbacks. Finally, as we will see in Part VIII, it is a cornerstone of any Continuous Integration environment.

In this section, we will look at the two most prominent open source version control tools on the market: CVS and Subversion.

Unit Testing

Unit testing is another important best practice of modern software development. Although testing is certainly not new in itself, unit testing, and practices such as Test-Driven Development, have been gaining popularity over recent years. Not only does proper unit testing help ensure that your code works; it also fosters cleaner, more modular, and better designed code. Automated unit testing takes this a step further. By simply integrating your unit tests into your standard build process, and running them automatically with every build, you can go a long way toward increasing the quality and reliability of your code.

Writing tests is good, but it is even better to be sure of what code you are actually testing. Test coverage tools help you check how much of your application is actually being executed during your unit tests. This in turn helps you identify untested code and improve the overall quality of your tests.

In this section, we will cover the latest tools in the unit testing domain, including JUnit 4 and TestNG, and see how these tests can be integrated smoothly into the build process. We also will look at how to verify test coverage with Cobertura, a powerful open source coverage tool.

Integration, Load, and Performance Testing

Unit testing is not the end of the story as far as testing goes. This section looks at other testing techniques such as integration, load and performance, and user interface testing. All of these are important, and all can benefit from being integrated into the build process.

In this section, we will see how to integrate performance tests into your unit tests, how to load-test your application, and how to automatically test web services, Swing interfaces, and web interfaces.

Quality Metrics Tools

It is important to be able to measure the quality of your code in objective terms. Code quality has a direct bearing on the number of bugs and the ease of maintenance later on. Code quality metrics can also be a good way to bring inexperienced developers up to speed with coding conventions and best practices. This section looks at a range of automated tools that measure different aspects of code quality, including CheckStyle, PMD, FindBugs, and Jupiter.

Technical Documentation Tools

A professional project needs professional documentation. A significant part of this documentation can (and should) be generated automatically, based on the source code and comments. This is the most reliable way to get consistently up-to-date technical documentation. This section describes tools that can help you generate good technical documentation without having to spend too much effort writing and maintaining it.

Issue Management Tools

We will look at that vital communication tool in the SDLC, the issue tracking system. Of course, issue tracking systems can be used by testers to raise bugs and by developers to document bug fixes. But they can also be used to help organize and document releases, to plan iterations, and to assign work tasks to team members.

There are literally hundreds of issue tracking systems out there, both open source and commercial. In this section, we will look at two of the more interesting open source solutions seen in the Java world. The first is Bugzilla, the original open source issue tracking system. The second is Trac, which excels by its excellent Subversion integration and its innovative project management and wiki features.

Continuous Integration Tools

Finally, we look at a tool to wrap it all up together under a single process. This is the proverbial "one tool to rule them all." This process is called continuous integration.

In software development, it is a common observation that the longer you wait to integrate your team's code, the harder it gets. Continuous Integration is based on the idea that you can greatly facilitate this process by committing small changes regularly, and then running automatic builds whenever code changes are committed. Whenever a developper commits new or modified code to the source code repository, the build server checks it out and runs a build. In the very least, this makes sure that the modifications compile correctly. However, why stop at simply checking that everything compiles? While you're at it, you might as well check that all the unit tests still run not forgetting, of course, the integration, performance, and user interface tests.

Indeed, virtually all of the tools and techniques that we have discussed above can benefit from being run automatically on a regular basis.

Although this sort of integration is certainly possible with a well-tailored shell script and a cron job, nowadays there are a lot of tools that can save you a great deal of time and effort in this area. In this section, we will be looking at some of the more interesting open source CI tools: Continuum, CruiseControl, LuntBuild, and Hudson.

Who Should Read This Book

This is, fundamentally, a techie book. It is a hands on tour of a wide range of tools, for people who like to get their hands dirty.

If you are a Java developer, these tools can help to improve your development practices, and make your life easier in the process. Lead developers, architects, and people interested in the wider picture will be able to glean from these pages some useful ideas about improving your project infrastructure and best practices. You will learn about the different build tools of the Java world. You will learn how to set up a version control server or a Continuous Integration server using open source tools. You will find tools to support your coding standards and design recommendations, and automatically generate high-quality technical documentation. You will find out how to get the most out of your unit and integration tests.

Readers are expected to have a basic knowledge of Java and XML. Many build servers run on Linux boxes, so there will be a mixture of Windows and Linux examples, when operating systems issues are discussed. No prior experience of any of the tools is required.

What This Book Doesn't Cover

This book cannot come close to covering all the good software tools on the market. Some aren't here for lack of space, or simply because I wasn't familiar enough with them to do them justice.

This book is limited to open source tools. This is not because there are not commercial tools in the software development lifecycle field: there are. Nor is it because these tools aren't worth considering for your project or organization; again, they may be. No, commercial tools are off limits for a simple question of scope (I did want to finish this book one day), and, to be fair to everyone, it would be hard to include one tool without including all the others.

Having said this, there are a few excellent commercial tools on the market which it would be a shame not to mention. These commercial tools are often of very high quality, with many innovative features. And, as in any field, competition is always an excellent driver for innovation.

Two organizations that deserve special mention in this area are Atlassian and JetBrains. Atlassian is behind the very popular JIRA issue tracking system. They also market Bamboo, an innovative Continuous Integration server, as well as a set of integrated tools such as Clover, a test coverage tool; FishEye, a tool that helps you to visualize the contents of your source code repository; and Crucible, a code review tool.

JetBrains is the author of the well-known and highly innovative Java IDE IntelliJ. Recently, JetBrains have also developed TeamCity, a next-generation Continuous Integration server that builds and tests code *before* it is committed to version control.

At the time of this writing, both of these companies offered free licencing arrangements for open source products.

Contributing Authors

This book was not written alone. Indeed, it has been a collaborative effort, with the direct and indirect participation of many people. The following are those who generously contributed their time and energy into providing valuable material for this book:

Brian Agnew

Brian Agnew is the founder and principal consultant with OOPS Consultancy Ltd, located in London, U.K. He holds a B.Eng. in Electrical and Electronic Engineering from Sheffield University. He advises and works with major financial houses and leading consultancies on a wide range of projects, including trading systems, network management infrastructures and grid-based architectures, working in Java mainly, C++ when he has to, and Perl when nothing else will do.

Brian contributed material on XMLTask, an Ant task that provides sophisticated XML manipulation.

Jettro Coenradie

Jettro Coenradie is a Java (enterprise) specialist living in the Netherlands. Jettro has been working in the ICT for about 10 years now. He likes to try out new frameworks that are focused on quality and productivity, so he is interested in tools like Luntbuild, Continuum and Maven. In addition, Jettro is a software architect focusing on the Spring framework. You can find more information about Jettro on his blog, *http://www.gridshore.nl/blog*.

Jettro contributed an article on integrating Maven with LuntBuild.

Keith Coughtrey

Keith gained a wealth of development experience at some of the U.K.'s largest companies before moving to New Zealand in 2004. He is currently leading development at First NZ Captial, New Zealand's largest stockbroker, where he is applying many of the tools and techniques espoused in this book.

Keith contributed material on Mylyn.

John Hurst

John Hurst is an experienced Java developer. an independent software developer, and a systems integrator working in the Wellington, New Zealand, area. John is an active member of the Java community, and plays a key role in the Wellington Java User Group.

John contributed the chapter on DBUnit.

Masoud Kalali

With more than eight years of experience, Masoud Kalali is a senior staff engineer at E-peyk. He is mostly responsible for the design and technology architecture of E-peyk enterprise framework, which is a service-oriented framework based on Java EE 5 and other open standards. Masoud's area of expertise are SOA and web services in addition to performance monitoring and management. Masoud is a contributor of several open source projects that are utilized in the E-peyk framework.

Masoud contributed material on SchemaSpy and on SoapUI.

Avneet Mangat

Avneet Mangat has six years' experience with Java/JEE. He is currently working as the Lead Developer at Active Health Partners, U.K. He is a Sun-certified programmer and web developer and is also Prince2-certified. He is also the lead developer of DBBrowser open source database browsing tool (*http://database browser.sourceforge.net/*). Outside interests are photography and traveling.

Avneet contributed material on setting up a Maven repository with Artifactory.

Eric Redmond

Eric Redmond has been involved in the Maven community for over two years as a user, contributor, speaker, author, and patch provider, as well as a founder of two professional education and consulting companies surrounding Maven and other aspects of large-scale application lifecycle management. He is currently enjoying some downtime, meaning a simple life as a Ruby (and Rails) developer, yet keeping

a keen eye on JRuby. He is the proprietor of Propellors Consulting and an active blogger at *http://blog.propellors.net*.

Eric contributed material on Maven 2 and Achiva.

Alex Ruiz

Alex Ruiz is a Software Engineer in the development tools organization at Oracle (*http://www.oracle.com*). Alex enjoys reading anything related to Java, testing, OOP, AOP, and concurrency, and has programming as his first love. Alex has spoken at JavaOne, JavaPolis, Desktop Matters, and SD West. He also has publications with IEEE Software, Dev2Dev, JavaWorld and Objective View. Before joining Oracle, Alex was a consultant for ThoughtWorks. Alex maintains a blog at *http://www.jroller.com/page/alexRuiz*.

Alex has contributed material on FEST.

Tim O'Brien also helped out with ideas for the introduction.

Technical Reviewers

The following people were brave enough to accept the task of formally reviewing the finished manuscript:

Nigel Charman

Nigel is a Java consultant living in Wellington, New Zealand, with a special interest in developer testing and code quality. In his spare time, he's currently working on the JiBX/WS web services framework and helping to organize the Wellington Java User Group.

Paul Duvall

Paul M. Duvall is the CTO of Stelligent Incorporated in Reston, VA—an Agile infrastructure consulting firm. He coauthored *Continuous Integration: Improving Software Quality and Reducing Risk* (Addison-Wesley, 2007), authors a series for IBM developer*Works* called "Automation for the People," and is a contributing author to the *No Fluff Just Stuff Anthology* (Pragmatic Programmers, 2007) and the "UML 2 Toolkit" (Wiley, 2003). He actively blogs on TestEarly.com and IntegrateButton.com.

Greg Ostravich

Greg Ostravich works for the Colorado Department of Transportation in Denver, Colorado, where he leverages the knowledge from the local Denver Java User Group and books such as *Java Power Tools* to implement best practices in software development for the State of Colorado. In addition to being a technical editor for *Java Power Tools*, he is currently the President of the Denver Java User Group and did the technical editing for JBoss at Work, No Fluff Just Anthologies (2006 and 2007), and *GIS for Web Developers* (2007). He also wrote reviews for *Pragmatic Unit Testing in Java with JUnit* and *Pragmatic Project Automation*, which can be found on the Denver Java User Group and The Server Side web sites.

Many other people also reviewed individual chapters or sections, including Cédric Beust, Keith Coughtrey, Richard Hancock, Jimmy Kemp, Gregor Lawson, Andrew McDowell, Brett Porter, Bill Ross, and Martin White.

Conventions

Most of the conventions used in this book should be fairly self-explanatory. Literal text such as file names, class names, and so on, is represented using `a fixed width font`. Commands intended to be executed on the command line are written in ***`constant width italic`***. Code listings are written like this:

```
<settings>
    <servers>
        <server>
            <id>organisation-internal</id>
            <username>admin</username>
            <password>password</password>
        </server>
    </servers>
</settings>
```

Sometimes, for the sake of brevity, I will use three dots ("...") to indicate that I've left out some stuff, as shown here:

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://artifactory.jfrog.org/xsd/1.0.0"
        xsi:schemaLocation="http://artifactory.jfrog.org/xsd/1.0.0
        http://www.jfrog.org/xsd/artifactory-v1_0_0.xsd">
    ...
    <remoteRepositories>
      <remoteRepository>
          <key>ibiblio</key>
          <handleReleases>true</handleReleases>
          <handleSnapshots>false</handleSnapshots>
          <excludesPattern>org/artifactory/**,org/jfrog/**</excludesPattern>
          <url>http://repo1.maven.org/maven2</url>
          <proxyRef>proxy1</proxyRef>
      </remoteRepository>
    </remoteRepositories>
    <proxies>
        <proxy>
            <key>proxy1</key>
            <host>proxyhost</host>
            <port>8080</port>
            <username>proxy</username>
            <password>secret</password>
        </proxy>
    </proxies>
</config>
```

When giving examples of commands executed from the command line, I usually use the Unix-style "$" to indicate the command-line prompt (or "#" for root access). The

Windows equivalent would be something like "C:>." Commands that are typed by are written *like this*. System output is written in normal type:

```
$ mvn compile
[INFO] Scanning for projects...
Downloading: http://buildserver:8080/artifactory/repo/org/apache/maven/wagon/wagon-
ssh-external/1.0-alpha-5/wagon-ssh-external-1.0-alpha-5.pom
5K downloaded
...
```

The commands themselves are generally the same under Unix and Windows, with the exception of the usual things such as class pathes. You might also see the occasional *ls* rather than *dir* in some of the examples.

When a command is split over several lines for readability, I use the Unix convention of putting a "\" at the end of each line except the last one. In Windows, you will have to put everything on one line (excluding the trailing "\" characters):

```
$ mvn deploy:deploy-file -DrepositoryId=organisation-internal \
  -Durl=http://buildserver:8080/artifactory/private-internal-repository \
  DgroupId=test -DartifactId=test -Dversion=1.1 -Dpackaging=jar -Dfile=target
  /test-1.1.jar
```

In a book like this, there are many occasions where screenshots come in handy. To save space and avoid distraction, all of the browser screenshots have been cropped to remove the window header and navigation bars. On the rare occasion where this has not been done, the URL is usually relevent to the discussion. Application windows such as IDEs are shown in full or partially, depending on the context.

Source Code

There are a lot of practical examples throughout this book. You can download the source code for these examples from the publisher's web site, or from the Java Power Tools web site (*http://www.javapowertools.com*). In general, the code is designed to illustrate how to use the various tools, and to make it possible for you to experiment with the tools. As a rule, it is not of production code quality, as production quality code tends to be more complex and project-specific.

About the Title

This book is part of the O'Reilly Power Tools series, which began with the illustrious Unix Power Tools back in 1993. It has no relation with the "Java Power Tools" library (*http://www.ccs.neu.edu/jpt/*), which is a software research project in the Java GUI field, conducted by College of Computer & Information Science at the Northeastern University, Boston, under the direction of Richard Rasala.