This page intentionally left blank
Preface ............................................................................................................................. xv

Acknowledgments ........................................................................................................ xix

Chapter 1: The Java SE 8 Stream Library ................................................................... 1
  1.1 From Iterating to Stream Operations .......................................................... 2
  1.2 Stream Creation .......................................................................................... 5
  1.3 The filter, map, and flatMap Methods .................................................... 9
  1.4 Extracting Substreams and Concatenating Streams .......................... 10
  1.5 Other Stream Transformations ............................................................ 11
  1.6 Simple Reductions ................................................................................... 12
  1.7 The Optional Type ................................................................................... 13
      1.7.1 How to Work with Optional Values ........................................... 14
      1.7.2 How Not to Work with Optional Values .................................. 15
      1.7.3 Creating Optional Values ......................................................... 16
      1.7.4 Composing Optional Value Functions with flatMap ............. 16
  1.8 Collecting Results .................................................................................... 19
  1.9 Collecting into Maps ................................................................................ 24
  1.10 Grouping and Partitioning ................................................................. 28
  1.11 Downstream Collectors .......................................................................... 29
  1.12 Reduction Operations ............................................................................ 33
  1.13 Primitive Type Streams ......................................................................... 36
  1.14 Parallel Streams ...................................................................................... 41

Chapter 2: Input and Output ..................................................................................... 47
  2.1 Input/Output Streams ............................................................................... 48
      2.1.1 Reading and Writing Bytes ...................................................... 48
      2.1.2 The Complete Stream Zoo ...................................................... 51
      2.1.3 Combining Input/Output Stream Filters ................................. 55
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2 Parsing an XML Document</td>
<td>149</td>
</tr>
<tr>
<td>3.3 Validating XML Documents</td>
<td>162</td>
</tr>
<tr>
<td>3.3.1 Document Type Definitions</td>
<td>163</td>
</tr>
<tr>
<td>3.3.2 XML Schema</td>
<td>172</td>
</tr>
<tr>
<td>3.3.3 A Practical Example</td>
<td>175</td>
</tr>
<tr>
<td>3.4 Locating Information with XPath</td>
<td>190</td>
</tr>
<tr>
<td>3.5 Using Namespaces</td>
<td>196</td>
</tr>
<tr>
<td>3.6 Streaming Parsers</td>
<td>199</td>
</tr>
<tr>
<td>3.6.1 Using the SAX Parser</td>
<td>199</td>
</tr>
<tr>
<td>3.6.2 Using the StAX Parser</td>
<td>205</td>
</tr>
<tr>
<td>3.7 Generating XML Documents</td>
<td>208</td>
</tr>
<tr>
<td>3.7.1 Documents without Namespaces</td>
<td>209</td>
</tr>
<tr>
<td>3.7.2 Documents with Namespaces</td>
<td>209</td>
</tr>
<tr>
<td>3.7.3 Writing Documents</td>
<td>210</td>
</tr>
<tr>
<td>3.7.4 An Example: Generating an SVG File</td>
<td>211</td>
</tr>
<tr>
<td>3.7.5 Writing an XML Document with StAX</td>
<td>214</td>
</tr>
<tr>
<td>3.8 XSL Transformations</td>
<td>222</td>
</tr>
<tr>
<td>Chapter 4: Networking</td>
<td>233</td>
</tr>
<tr>
<td>4.1 Connecting to a Server</td>
<td>233</td>
</tr>
<tr>
<td>4.1.1 Using Telnet</td>
<td>233</td>
</tr>
<tr>
<td>4.1.2 Connecting to a Server with Java</td>
<td>236</td>
</tr>
<tr>
<td>4.1.3 Socket Timeouts</td>
<td>238</td>
</tr>
<tr>
<td>4.1.4 Internet Addresses</td>
<td>239</td>
</tr>
<tr>
<td>4.2 Implementing Servers</td>
<td>241</td>
</tr>
<tr>
<td>4.2.1 Server Sockets</td>
<td>242</td>
</tr>
<tr>
<td>4.2.2 Serving Multiple Clients</td>
<td>245</td>
</tr>
<tr>
<td>4.2.3 Half-Close</td>
<td>249</td>
</tr>
<tr>
<td>4.3 Interruptible Sockets</td>
<td>250</td>
</tr>
<tr>
<td>4.4 Getting Web Data</td>
<td>257</td>
</tr>
<tr>
<td>4.4.1 URLs and URIs</td>
<td>257</td>
</tr>
<tr>
<td>4.4.2 Using a URLConnection to Retrieve Information</td>
<td>259</td>
</tr>
<tr>
<td>4.4.3 Posting Form Data</td>
<td>267</td>
</tr>
<tr>
<td>4.5 Sending E-Mail</td>
<td>277</td>
</tr>
</tbody>
</table>
9.2 Security Managers and Permissions ..................................................... 509
  9.2.1 Permission Checking .............................................................. 509
  9.2.2 Java Platform Security ............................................................. 510
  9.2.3 Security Policy Files ............................................................... 514
  9.2.4 Custom Permissions ............................................................... 522
  9.2.5 Implementation of a Permission Class ................................... 524

9.3 User Authentication ................................................................. 530
  9.3.1 The JAAS Framework ........................................................... 531
  9.3.2 JAAS Login Modules ............................................................ 537

9.4 Digital Signatures ................................................................. 546
  9.4.1 Message Digests ................................................................. 547
  9.4.2 Message Signing ................................................................. 550
  9.4.3 Verifying a Signature ............................................................ 553
  9.4.4 The Authentication Problem ................................................ 556
  9.4.5 Certificate Signing ............................................................... 558
  9.4.6 Certificate Requests ............................................................ 560
  9.4.7 Code Signing ....................................................................... 561

9.5 Encryption ............................................................................... 567
  9.5.1 Symmetric Ciphers ............................................................... 567
  9.5.2 Key Generation ................................................................. 569
  9.5.3 Cipher Streams ................................................................. 574
  9.5.4 Public Key Ciphers ............................................................. 575

Chapter 10: Advanced Swing .......................................................... 581

10.1 Lists ...................................................................................... 582
  10.1.1 The JList Component .......................................................... 582
  10.1.2 List Models ......................................................................... 588
  10.1.3 Inserting and Removing Values ................................. 593
  10.1.4 Rendering Values ............................................................ 595

10.2 Tables .................................................................................. 599
  10.2.1 A Simple Table ................................................................. 600
  10.2.2 Table Models ................................................................. 604
  10.2.3 Working with Rows and Columns .................. 608
    10.2.3.1 Column Classes .......................................................... 609
    10.2.3.2 Accessing Table Columns .................................. 610
    10.2.3.3 Resizing Columns ...................................................... 611
10.2.3.4 Resizing Rows ............................................................ 612
10.2.3.5 Selecting Rows, Columns, and Cells ....................... 612
10.2.3.6 Sorting Rows .............................................................. 614
10.2.3.7 Filtering Rows ......................................................... 615
10.2.3.8 Hiding and Displaying Columns ............................ 617
10.2.4 Cell Rendering and Editing ............................................. 626
  10.2.4.1 Rendering Cells ................................................... 626
  10.2.4.2 Rendering the Header ......................................... 627
  10.2.4.3 Editing Cells ....................................................... 628
  10.2.4.4 Custom Editors .................................................. 629
10.3 Trees ......................................................................................... 639
  10.3.1 Simple Trees ............................................................. 640
  10.3.2 Editing Trees and Tree Paths .................................... 650
  10.3.3 Node Enumeration ................................................... 659
  10.3.4 Rendering Nodes ...................................................... 661
  10.3.5 Listening to Tree Events ........................................... 664
  10.3.6 Custom Tree Models .................................................. 671
10.4 Text Components ................................................................. 681
  10.4.1 Change Tracking in Text Components ...................... 682
  10.4.2 Formatted Input Fields .............................................. 685
    10.4.2.1 Integer Input ..................................................... 686
    10.4.2.2 Behavior on Loss of Focus ................................. 687
    10.4.2.3 Filters ............................................................ 688
    10.4.2.4 Verifiers .......................................................... 690
    10.4.2.5 Other Standard Formatters ............................... 691
    10.4.2.6 Custom Formatters .......................................... 693
  10.4.3 The JSpinner Component ............................................ 703
  10.4.4 Displaying HTML with the JEditorPane .................. 712
10.5 Progress Indicators .............................................................. 719
  10.5.1 Progress Bars .......................................................... 719
  10.5.2 Progress Monitors ................................................... 722
  10.5.3 Monitoring the Progress of Input Streams .................. 726
10.6 Component Organizers and Decorators ............................... 731
  10.6.1 Split Panes ............................................................ 732
  10.6.2 Tabbed Panes ........................................................ 735
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.6.3</td>
<td>Desktop Panes and Internal Frames</td>
<td>741</td>
</tr>
<tr>
<td>10.6.3.1</td>
<td>Displaying Internal Frames</td>
<td>741</td>
</tr>
<tr>
<td>10.6.3.2</td>
<td>Cascading and Tiling</td>
<td>744</td>
</tr>
<tr>
<td>10.6.3.3</td>
<td>Vetoing Property Settings</td>
<td>748</td>
</tr>
<tr>
<td>10.6.3.4</td>
<td>Dialogs in Internal Frames</td>
<td>750</td>
</tr>
<tr>
<td>10.6.3.5</td>
<td>Outline Dragging</td>
<td>751</td>
</tr>
<tr>
<td>10.6.4</td>
<td>Layers</td>
<td>760</td>
</tr>
</tbody>
</table>

**Chapter 11: Advanced AWT**

- 11.1 The Rendering Pipeline | 766 |
- 11.2 Shapes |
  - 11.2.1 The Shape Class Hierarchy | 769 |
  - 11.2.2 Using the Shape Classes | 772 |
- 11.3 Areas | 786 |
- 11.4 Strokes | 788 |
- 11.5 Paint | 797 |
- 11.6 Coordinate Transformations | 799 |
- 11.7 Clipping | 805 |
- 11.8 Transparency and Composition | 807 |
- 11.9 Rendering Hints | 817 |
- 11.10 Readers and Writers for Images |
  - 11.10.1 Obtaining Readers and Writers for Image File Types | 824 |
  - 11.10.2 Reading and Writing Files with Multiple Images | 825 |
- 11.11 Image Manipulation | 834 |
  - 11.11.1 Constructing Raster Images | 835 |
  - 11.11.2 Filtering Images | 842 |
- 11.12 Printing |
  - 11.12.1 Graphics Printing | 852 |
  - 11.12.2 Multiple-Page Printing | 862 |
  - 11.12.3 Print Preview | 864 |
  - 11.12.4 Print Services | 874 |
  - 11.12.5 Stream Print Services | 878 |
  - 11.12.6 Printing Attributes | 879 |
- 11.13 The Clipboard | 887 |
  - 11.13.1 Classes and Interfaces for Data Transfer | 888 |
  - 11.13.2 Transferring Text | 888 |
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.13.3 The Transferable Interface and Data Flavors</td>
<td>892</td>
</tr>
<tr>
<td>11.13.4 Building an Image Transferable</td>
<td>894</td>
</tr>
<tr>
<td>11.13.5 Transferring Java Objects via the System Clipboard</td>
<td>898</td>
</tr>
<tr>
<td>11.13.6 Using a Local Clipboard to Transfer Object References</td>
<td>902</td>
</tr>
<tr>
<td>11.14 Drag and Drop</td>
<td>903</td>
</tr>
<tr>
<td>11.14.1 Data Transfer Support in Swing</td>
<td>904</td>
</tr>
<tr>
<td>11.14.2 Drag Sources</td>
<td>909</td>
</tr>
<tr>
<td>11.14.3 Drop Targets</td>
<td>912</td>
</tr>
<tr>
<td>11.15 Platform Integration</td>
<td>921</td>
</tr>
<tr>
<td>11.15.1 Splash Screens</td>
<td>921</td>
</tr>
<tr>
<td>11.15.2 Launching Desktop Applications</td>
<td>927</td>
</tr>
<tr>
<td>11.15.3 The System Tray</td>
<td>932</td>
</tr>
<tr>
<td><strong>Chapter 12: Native Methods</strong></td>
<td>939</td>
</tr>
<tr>
<td>12.1 Calling a C Function from a Java Program</td>
<td>940</td>
</tr>
<tr>
<td>12.2 Numeric Parameters and Return Values</td>
<td>947</td>
</tr>
<tr>
<td>12.3 String Parameters</td>
<td>949</td>
</tr>
<tr>
<td>12.4 Accessing Fields</td>
<td>956</td>
</tr>
<tr>
<td>12.4.1 Accessing Instance Fields</td>
<td>956</td>
</tr>
<tr>
<td>12.4.2 Accessing Static Fields</td>
<td>960</td>
</tr>
<tr>
<td>12.5 Encoding Signatures</td>
<td>961</td>
</tr>
<tr>
<td>12.6 Calling Java Methods</td>
<td>963</td>
</tr>
<tr>
<td>12.6.1 Instance Methods</td>
<td>963</td>
</tr>
<tr>
<td>12.6.2 Static Methods</td>
<td>964</td>
</tr>
<tr>
<td>12.6.3 Constructors</td>
<td>965</td>
</tr>
<tr>
<td>12.6.4 Alternative Method Invocations</td>
<td>966</td>
</tr>
<tr>
<td>12.7 Accessing Array Elements</td>
<td>970</td>
</tr>
<tr>
<td>12.8 Handling Errors</td>
<td>974</td>
</tr>
<tr>
<td>12.9 Using the Invocation API</td>
<td>980</td>
</tr>
<tr>
<td>12.10 A Complete Example: Accessing the Windows Registry</td>
<td>985</td>
</tr>
<tr>
<td>12.10.1 Overview of the Windows Registry</td>
<td>985</td>
</tr>
<tr>
<td>12.10.2 A Java Platform Interface for Accessing the Registry</td>
<td>987</td>
</tr>
<tr>
<td>12.10.3 Implementation of Registry Access Functions as Native Methods</td>
<td>988</td>
</tr>
</tbody>
</table>

**Index** ...........................................................................................................1002
Preface

To the Reader

The book you have in your hands is the second volume of the tenth edition of Core Java®, fully updated for Java SE 8. The first volume covers the essential features of the language; this volume deals with the advanced topics that a programmer needs to know for professional software development. Thus, as with the first volume and the previous editions of this book, we are still targeting programmers who want to put Java technology to work in real projects.

As is the case with any book, errors and inaccuracies are inevitable. Should you find any in this book, we would very much like to hear about them. Of course, we would prefer to hear about them only once. For this reason, we have put up a web site at http://horstmann.com/corejava with a FAQ, bug fixes, and workarounds. Strategically placed at the end of the bug report web page (to encourage you to read the previous reports) is a form that you can use to report bugs or problems and to send suggestions for improvements to future editions.

About This Book

The chapters in this book are, for the most part, independent of each other. You should be able to delve into whatever topic interests you the most and read the chapters in any order.

In Chapter 1, you will learn all about the Java 8 stream library that brings a modern flavor to processing data, by specifying what you want without describing in detail how the result should be obtained. This allows the stream library to focus on an optimal evaluation strategy, which is particularly advantageous for optimizing concurrent computations.

The topic of Chapter 2 is input and output handling (I/O). In Java, all input and output is handled through input/output streams. These streams (not to be confused with those in Chapter 1) let you deal, in a uniform manner, with communications among various sources of data, such as files, network connections, or memory blocks. We include detailed coverage of the reader and writer classes that make it easy to deal with Unicode. We show you what goes on under the
hood when you use the object serialization mechanism, which makes saving and loading objects easy and convenient. We then move on to regular expressions and working with files and paths.

**Chapter 3** covers XML. We show you how to parse XML files, how to generate XML, and how to use XSL transformations. As a useful example, we show you how to specify the layout of a Swing form in XML. We also discuss the XPath API, which makes “finding needles in XML haystacks” much easier.

**Chapter 4** covers the networking API. Java makes it phenomenally easy to do complex network programming. We show you how to make network connections to servers, how to implement your own servers, and how to make HTTP connections.

**Chapter 5** covers database programming. The main focus is on JDBC, the Java database connectivity API that lets Java programs connect to relational databases. We show you how to write useful programs to handle realistic database chores, using a core subset of the JDBC API. (A complete treatment of the JDBC API would require a book almost as big as this one.) We finish the chapter with a brief introduction into hierarchical databases and discuss JNDI (the Java Naming and Directory Interface) and LDAP (the Lightweight Directory Access Protocol).

Java had two prior attempts at libraries for handling date and time. The third one is the charm in Java 8. In **Chapter 6**, you will learn how to deal with the complexities of calendars and time zones, using the new date and time library.

**Chapter 7** discusses a feature that we believe can only grow in importance: internationalization. The Java programming language is one of the few languages designed from the start to handle Unicode, but the internationalization support in the Java platform goes much further. As a result, you can internationalize Java applications so that they cross not only platforms but country boundaries as well. For example, we show you how to write a retirement calculator that uses either English, German, or Chinese languages.

**Chapter 8** discusses three techniques for processing code. The scripting and compiler APIs allow your program to call code in scripting languages such as JavaScript or Groovy, and to compile Java code. Annotations allow you to add arbitrary information (sometimes called metadata) to a Java program. We show you how annotation processors can harvest these annotations at the source or class file level, and how annotations can be used to influence the behavior of classes at runtime. Annotations are only useful with tools, and we hope that our discussion will help you select useful annotation processing tools for your needs.

**Chapter 9** takes up the Java security model. The Java platform was designed from the ground up to be secure, and this chapter takes you under the hood to see how this design is implemented. We show you how to write your own class loaders...
and security managers for special-purpose applications. Then, we take up the security API that allows for such important features as message and code signing, authorization and authentication, and encryption. We conclude with examples that use the AES and RSA encryption algorithms.

Chapter 10 contains all the Swing material that didn’t make it into Volume I, especially the important but complex tree and table components. We show the basic uses of editor panes, the Java implementation of a “multiple document” interface, progress indicators used in multithreaded programs, and “desktop integration features” such as splash screens and support for the system tray. Again, we focus on the most useful constructs that you are likely to encounter in practical programming because an encyclopedic coverage of the entire Swing library would fill several volumes and would only be of interest to dedicated taxonomists.

Chapter 11 covers the Java 2D API, which you can use to create realistic drawings and special effects. The chapter also covers some advanced features of the AWT (Abstract Windowing Toolkit) that seemed too specialized for coverage in Volume I but should, nonetheless, be part of every programmer’s toolkit. These features include printing and the APIs for cut-and-paste and drag-and-drop.

Chapter 12 takes up native methods, which let you call methods written for a specific machine such as the Microsoft Windows API. Obviously, this feature is controversial: Use native methods, and the cross-platform nature of Java vanishes. Nonetheless, every serious programmer writing Java applications for specific platforms needs to know these techniques. At times, you need to turn to the operating system’s API for your target platform when you interact with a device or service that is not supported by Java. We illustrate this by showing you how to access the registry API in Windows from a Java program.

As always, all chapters have been completely revised for the latest version of Java. Outdated material has been removed, and the new APIs of Java SE 8 are covered in detail.

Conventions

As is common in many computer books, we use monospace type to represent computer code.

NOTE: Notes are tagged with “note” icons that look like this.
Java comes with a large programming library, or Application Programming Interface (API). When using an API call for the first time, we add a short summary description at the end of the section. These descriptions are a bit more informal but, we hope, also a little more informative than those in the official online API documentation. The names of interfaces are in italics, just like in the official documentation. The number after a class, interface, or method name is the JDK version in which the feature was introduced.

Application Programming Interface 1.2

Programs whose source code is included in the companion code for this book are listed as examples; for instance,

Listing 1.1  ScriptTest.java

You can download the companion code from http://horstmann.com/corejava.
Acknowledgments

Writing a book is always a monumental effort, and rewriting doesn’t seem to be much easier, especially with such a rapid rate of change in Java technology. Making a book a reality takes many dedicated people, and it is my great pleasure to acknowledge the contributions of the entire Core Java team.

A large number of individuals at Prentice Hall provided valuable assistance, but they managed to stay behind the scenes. I’d like them all to know how much I appreciate their efforts. As always, my warm thanks go to my editor, Greg Doench, for steering the book through the writing and production process, and for allowing me to be blissfully unaware of the existence of all those folks behind the scenes. I am very grateful to Julie Nahil for production support, and to Dmitry Kirsanov and Alina Kirsanova for copyediting and typesetting the manuscript.

Thanks to the many readers of earlier editions who reported embarrassing errors and made lots of thoughtful suggestions for improvement. I am particularly grateful to the excellent reviewing team that went over the manuscript with an amazing eye for detail and saved me from many more embarrassing errors.

Reviewers of this and earlier editions include Chuck Allison (Contributing Editor, C/C++ Users Journal), Lance Anderson (Oracle), Alec Beaton (PointBase, Inc.), Cliff Berg (iSavvix Corporation), Joshua Bloch, David Brown, Corky Cartwright, Frank Cohen (PushToTest), Chris Crane (devXsolution), Dr. Nicholas J. De Lillo (Manhattan College), Rakesh Dhoopar (Oracle), Robert Evans (Senior Staff, The Johns Hopkins University Applied Physics Lab), David Geary (Sabreware), Jim Gish (Oracle), Brian Goetz (Oracle), Angela Gordon, Dan Gordon, Rob Gordon, John Gray (University of Hartford), Cameron Gregory (olabs.com), Steve Haines, Marty Hall (The Johns Hopkins University Applied Physics Lab), Vincent Hardy, Dan Harkey (San Jose State University), William Higgins (IBM), Vladimir Ivanovic (PointBase), Jerry Jackson (ChannelPoint Software), Tim Kimmet (Preview Systems), Chris Laffra, Charlie Lai, Angelika Langer, Doug Langston, Hang Lau (McGill University), Mark Lawrence, Doug Lea (SUNY Oswego), Gregory Longshore, Bob Lynch (Lynch Associates), Philip Milne (consultant), Mark Morrissey (The Oregon Graduate Institute), Mahesh Neelakanta (Florida Atlantic University), Hao Pham, Paul Philion, Blake Ragsdell, Ylber Ramadani (Ryerson University), Stuart Reges (University of Arizona), Simon Ritter, Rich Rosen (Interactive Data Corporation), Peter Sanders (ESSI University, Nice, France), Dr. Paul Sanghera (San Jose State University and Brooks College), Paul Sevinc (Teamup AG), Yoshiki Shabata, Devang Shah, Richard Slywczak (NASA/Glenn
Research Center), Bradley A. Smith, Steven Stelting, Christopher Taylor, Luke Taylor (Valtech), George Thiruvathukal, Kim Topley (author of Core JFC, Second Edition), Janet Traub, Paul Tyma (consultant), Christian Ullenboom, Peter van der Linden, Burt Walsh, Joe Wang (Oracle), and Dan Xu (Oracle).

Cay Horstmann
San Francisco, California
September 2016
This chapter introduces three techniques for processing code. The scripting API lets you invoke code in a scripting language such as JavaScript or Groovy. You can use the compiler API when you want to compile Java code inside your application. Annotation processors operate on Java source or class files that contain annotations. As you will see, there are many applications for annotation processing, ranging from simple diagnostics to “bytecode engineering”—the insertion of bytecodes into class files or even running programs.
8.1 Scripting for the Java Platform

A scripting language is a language that avoids the usual edit/compile/link/run cycle by interpreting the program text at runtime. Scripting languages have a number of advantages:

- Rapid turnaround, encouraging experimentation
- Changing the behavior of a running program
- Enabling customization by program users

On the other hand, most scripting languages lack features that are beneficial for programming complex applications, such as strong typing, encapsulation, and modularity.

It is therefore tempting to combine the advantages of scripting and traditional languages. The scripting API lets you do just that for the Java platform. It enables you to invoke scripts written in JavaScript, Groovy, Ruby, and even exotic languages such as Scheme and Haskell, from a Java program. For example, the Renjin project (www.renjin.org) provides a Java implementation of the R programming language, which is commonly used for statistical programming, together with an “engine” of the scripting API.

In the following sections, we’ll show you how to select an engine for a particular language, how to execute scripts, and how to make use of advanced features that some scripting engines offer.

8.1.1 Getting a Scripting Engine

A scripting engine is a library that can execute scripts in a particular language. When the virtual machine starts, it discovers the available scripting engines. To enumerate them, construct a ScriptEngineManager and invoke the getEngineFactories method. You can ask each engine factory for the supported engine names, MIME types, and file extensions. Table 8.1 shows typical values.

Usually, you know which engine you need, and you can simply request it by name, MIME type, or extension. For example:

```java
ScriptEngine engine = manager.getEngineByName("nashorn");
```

Java SE 8 includes a version of Nashorn, a JavaScript interpreter developed by Oracle. You can add more languages by providing the necessary JAR files on the class path.
### Table 8.1 Properties of Scripting Engine Factories

<table>
<thead>
<tr>
<th>Engine</th>
<th>Names</th>
<th>MIME types</th>
<th>Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nashorn (included with Java SE)</td>
<td>nashorn, Nashorn, js, JS, JavaScript, javascript, ECMAScript, ecmascript</td>
<td>application/javascript, application/ecmascript, text/javascript, text/ecmascript</td>
<td>js</td>
</tr>
<tr>
<td>Groovy</td>
<td>groovy</td>
<td>None</td>
<td>groovy</td>
</tr>
<tr>
<td>Renjin</td>
<td>Renjin</td>
<td>text/x-R</td>
<td>R, r, S, s</td>
</tr>
<tr>
<td>SISC Scheme</td>
<td>sisc</td>
<td>None</td>
<td>scheme, sisc</td>
</tr>
</tbody>
</table>

#### javax.script.ScriptEngineManager

- `List<ScriptEngineFactory> getEngineFactories()` gets a list of all discovered engine factories.
- `ScriptEngine getEngineByName(String name)`
- `ScriptEngine getEngineByExtension(String extension)`
- `ScriptEngine getEngineByMimeType(String mimeType)` gets the script engine with the given name, script file extension, or MIME type.

#### javax.script.ScriptEngineFactory

- `List<String> getNames()`
- `List<String> getExtensions()`
- `List<String> getMimeTypes()` gets the names, script file extensions, and MIME types under which this factory is known.

### 8.1.2 Script Evaluation and Bindings

Once you have an engine, you can call a script simply by invoking

```java
Object result = engine.eval(scriptString);
```

If the script is stored in a file, open a `Reader` and call

```java
Object result = engine.eval(reader);
```
You can invoke multiple scripts on the same engine. If one script defines variables, functions, or classes, most scripting engines retain the definitions for later use. For example,

```java
engine.eval("n = 1728");
Object result = engine.eval("n + 1");
```

will return 1729.

**NOTE:** To find out whether it is safe to concurrently execute scripts in multiple threads, call

```java
Object param = factory.getParameter("THREADING");
```

The returned value is one of the following:

- `null`: Concurrent execution is not safe.
- "MULTITHREADED": Concurrent execution is safe. Effects from one thread might be visible from another thread.
- "THREAD-ISOLATED": In addition to "MULTITHREADED", different variable bindings are maintained for each thread.
- "STATELESS": In addition to "THREAD-ISOLATED", scripts do not alter variable bindings.

You will often want to add variable bindings to the engine. A binding consists of a name and an associated Java object. For example, consider these statements:

```java
engine.put("k", 1728);
Object result = engine.eval("k + 1");
```

The script code reads the definition of `k` from the bindings in the “engine scope.” This is particularly important because most scripting languages can access Java objects, often with a syntax that is simpler than the Java syntax. For example,

```java
engine.put("b", new JButton());
engine.eval("b.text = 'Ok'"Object result = engine.eval("n = 1728");
Object result = engine.get("n");
```

Conversely, you can retrieve variables that were bound by scripting statements:

In addition to the engine scope, there is also a global scope. Any bindings that you add to the `ScriptEngineManager` are visible to all engines.
Instead of adding bindings to the engine or global scope, you can collect them in an object of type `Bindings` and pass it to the `eval` method:

```java
Bindings scope = engine.createBindings();
scope.put("b", new JButton());
engine.eval(scriptString, scope);
```

This is useful if a set of bindings should not persist for future calls to the `eval` method.

---

**NOTE:** You might want to have scopes other than the engine and global scopes. For example, a web container might need request and session scopes. However, then you are on your own. You will need to write a class that implements the `ScriptContext` interface, managing a collection of scopes. Each scope is identified by an integer number, and scopes with lower numbers should be searched first. (The standard library provides a `SimpleScriptContext` class, but it only holds global and engine scopes.)

---

**javax.script.ScriptEngine**

- `Object eval(String script)`
- `Object eval(Reader reader)`
- `Object eval(String script, Bindings bindings)`
- `Object eval(Reader reader, Bindings bindings)`
  
  evaluates the script given by the string or reader, subject to the given bindings.

- `Object get(String key)`
- `void put(String key, Object value)`
  
  gets or puts a binding in the engine scope.

- `Bindings createBindings()`
  
  creates an empty `Bindings` object suitable for this engine.

---

**javax.script.ScriptEngineManager**

- `Object get(String key)`
- `void put(String key, Object value)`
  
  gets or puts a binding in the global scope.
8.1.3 Redirecting Input and Output

You can redirect the standard input and output of a script by calling the `setReader` and `setWriter` methods of the script context. For example,

```java
StringWriter writer = new StringWriter();
engine.getContext().setWriter(new PrintWriter(writer, true));
```

Any output written with the JavaScript `print` or `println` functions is sent to `writer`. The `setReader` and `setWriter` methods only affect the scripting engine’s standard input and output sources. For example, if you execute the JavaScript code

```javascript
println("Hello");
java.lang.System.out.println("World");
```

only the first output is redirected.

The Nashorn engine does not have the notion of a standard input source. Calling `setReader` has no effect.

```java
javax.script.ScriptEngine
```

- `ScriptContext getContext()`
  - gets the default script context for this engine.

```java
javax.script.ScriptContext
```

- `Reader getReader()`
- `void setReader(Reader reader)`
- `Writer getWriter()`
- `void setWriter(Writer writer)`
- `Writer getErrorWriter()`
- `void setErrorWriter(Writer writer)`
  - gets or sets the reader for input or writer for normal or error output.
8.1.4 Calling Scripting Functions and Methods

With many script engines, you can invoke a function in the scripting language without having to evaluate the actual script code. This is useful if you allow users to implement a service in a scripting language of their choice.

The script engines that offer this functionality implement the `Invocable` interface. In particular, the Nashorn engine implements `Invocable`.

To call a function, call the `invokeFunction` method with the function name, followed by the function parameters:

```java
// Define greet function in JavaScript
engine.eval("function greet(how, whom) { return how + ', ' + whom + '!' }");

// Call the function with arguments "Hello", "World"
result = ((Invocable) engine).invokeFunction("greet", "Hello", "World");
```

If the scripting language is object-oriented, call `invokeMethod`:

```java
// Define Greeter class in JavaScript
engine.eval("function Greeter(how) { this.how = how }");
engine.eval("Greeter.prototype.welcome = 
 + " function(whom) { return this.how + ', ' + whom + '!' }");

// Construct an instance
Object yo = engine.eval("new Greeter('Yo')");

// Call the welcome method on the instance
result = ((Invocable) engine).invokeMethod(yo, "welcome", "World");
```

**NOTE:** For more information on how to define classes in JavaScript, see *JavaScript—The Good Parts* by Douglas Crockford (O'Reilly, 2008).

**NOTE:** If the script engine does not implement the `Invocable` interface, you might still be able to call a method in a language-independent way. The `getMethodCallSyntax` method of the `ScriptEngineFactory` interface produces a string that you can pass to the `eval` method. However, all method parameters must be bound to names, whereas `invokeMethod` can be called with arbitrary values.

You can go a step further and ask the scripting engine to implement a Java interface. Then you can call scripting functions and methods with the Java method call syntax.
The details depend on the scripting engine, but typically you need to supply a function for each method of the interface. For example, consider a Java interface:

```java
public interface Greeter {
    String welcome(String whom);
}
```

If you define a global function with the same name in Nashorn, you can call it through this interface:

```javascript
// Define welcome function in JavaScript
engine.eval("function welcome(whom) { return 'Hello, ' + whom + '!' }");

// Get a Java object and call a Java method
Greeter g = ((Invocable) engine).getInterface(Greeter.class);
result = g.welcome("World");
```

In an object-oriented scripting language, you can access a script class through a matching Java interface. For example, here is how to call an object of the JavaScript SimpleGreeter class with Java syntax:

```java
Greeter g = ((Invocable) engine).getInterface(yo, Greeter.class);
result = g.welcome("World");
```

In summary, the `Invocable` interface is useful if you want to call scripting code from Java without worrying about the scripting language syntax.

---

**javax.script.Invocable**

- `Object invokeFunction(String name, Object... parameters)`
- `Object invokeMethod(Object implicitParameter, String name, Object... explicitParameters)`
  - Invokes the function or method with the given name, passing the given parameters.
- `<T> T getInterface(Class<T> iface)`
  - Returns an implementation of the given interface, implementing the methods with functions in the scripting engine.
- `<T> T getInterface(Object implicitParameter, Class<T> iface)`
  - Returns an implementation of the given interface, implementing the methods with the methods of the given object.
8.1.5 Compiling a Script

Some scripting engines can compile scripting code into an intermediate form for efficient execution. Those engines implement the Compilable interface. The following example shows how to compile and evaluate code contained in a script file:

```java
Reader reader = new FileReader("myscript.js");
CompiledScript script = null;
if (engine implements Compilable)
  script = ((Compilable) engine).compile(reader);
```

Once the script is compiled, you can execute it. The following code executes the compiled script if compilation was successful, or the original script if the engine didn’t support compilation:

```java
if (script != null)
  script.eval();
else
  engine.eval(reader);
```

Of course, it only makes sense to compile a script if you need to execute it repeatedly.

### javax.script.Compilable

- CompiledScript compile(String script)
- CompiledScript compile(Reader reader)

  compiles the script given by a string or reader.

### javax.script.CompiledScript

- Object eval()
- Object eval(Bindings bindings)

  evaluates this script.

8.1.6 An Example: Scripting GUI Events

To illustrate the scripting API, we will write a sample program that allows users to specify event handlers in a scripting language of their choice.

Have a look at the program in Listing 8.1 that adds scripting to an arbitrary frame class. By default it reads the ButtonFrame class in Listing 8.2, which is similar to the event handling demo in Volume I, with two differences:
• Each component has its name property set.
• There are no event handlers.

The event handlers are defined in a property file. Each property definition has the form

    componentName.eventName = scriptCode

For example, if you choose to use JavaScript, supply the event handlers in a file js.properties, like this:

    yellowButton.action=panel.background = java.awt.Color.YELLOW
    blueButton.action=panel.background = java.awt.Color.BLUE
    redButton.action=panel.background = java.awt.Color.RED

The companion code also has files for Groovy, R, and SISC Scheme.

The program starts by loading an engine for the language specified on the command line. If no language is specified, we use JavaScript.

We then process a script init.language if it is present. This is useful for the R and Scheme languages, which need some initializations that we did not want to include in every event handler script.

Next, we recursively traverse all child components and add the bindings (name, object) into a map of components. Then we add the bindings to the engine.

Next, we read the file language.properties. For each property, we synthesize an event handler proxy that causes the script code to be executed. The details are a bit technical. You might want to read the section on proxies in Volume I, Chapter 6, if you want to follow the implementation in detail. The essential part, however, is that each event handler calls

    engine.eval(scriptCode);

Let us look at the yellowButton in more detail. When the line

    yellowButton.action=panel.background = java.awt.Color.YELLOW

is processed, we find the JButton component with the name "yellowButton". We then attach an ActionListener with an actionPerformed method that executes the script

    panel.background = java.awt.Color.YELLOW

if the scripting is done with Nashorn.

The engine contains a binding that binds the name "panel" to the JPanel object. When the event occurs, the setBackground method of the panel is executed, and the color changes.

You can run this program with the JavaScript event handlers simply by executing
For the Groovy handlers, use

```
java -classpath .:groovy/lib/\* ScriptTest groovy
```

Here, `groovy` is the directory into which you installed Groovy.

For the Renjin implementation of R, include the JAR files for Renjin Studio and the Renjin script engine on the classpath. Both are available at www.renjin.org/downloads.html.

To try out Scheme, download SISC Scheme from http://sisc-scheme.org and run

```
java -classpath .:sisc/:jsr223-engines/scheme/build/scheme-engine.jar ScriptTest scheme
```

where `sisc` is the installation directory for SISC Scheme and `jsr223-engines` is the directory that contains the engine adapters from http://java.net/projects/scripting.

This application demonstrates how to use scripting for Java GUI programming. One could go a step further and describe the GUI with an XML file, as you have seen in Chapter 3. Then our program would become an interpreter for GUIs that have visual presentation defined by XML and behavior defined by a scripting language. Note the similarity to a dynamic HTML page or a dynamic server-side scripting environment.

```
package script;

import java.awt.*;
import java.beans.*;
import java.io.*;
import java.lang.reflect.*;
import java.util.*;
import javax.script.*;
import javax.swing.*;

/**
 * @version 1.02 2016-05-10
 * @author Cay Horstmann
 */
public class ScriptTest
{
    public static void main(String[] args)
    {
        EventQueue.invokeLater(() ->
        {
            // (Continues)
```

Listing 8.1 script/ScriptTest.java
try {
    ScriptEngineManager manager = new ScriptEngineManager();
    String language;
    if (args.length == 0) {
        System.out.println("Available factories: ");
        for (ScriptEngineFactory factory : manager.getEngineFactories())
            System.out.println(factory.getEngineName());
        language = "nashorn";
    } else language = args[0];

    final ScriptEngine engine = manager.getEngineByName(language);
    if (engine == null) {
        System.err.println("No engine for " + language);
        System.exit(1);
    }

    final String frameClassName = args.length < 2 ? "buttons1.ButtonFrame" : args[1];
    JFrame frame = (JFrame) Class.forName(frameClassName).newInstance();
    InputStream in = frame.getClass().getResourceAsStream("init." + language);
    if (in != null) engine.eval(new InputStreamReader(in));
    Map<String, Component> components = new HashMap<>();
    getComponentBindings(frame, components);
    components.forEach((name, c) -> engine.put(name, c));

    final Properties events = new Properties();
    in = frame.getClass().getResourceAsStream(language + ".properties");
    events.load(in);

    for (final Object e : events.keySet()) {
        String[] s = ((String) e).split("\./");
        addListener(s[0], s[1], (String) events.get(e), engine, components);
    }
    frame.setTitle("ScriptTest");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
} catch (ReflectiveOperationException | IOException | ScriptException | IntrospectionException ex) {
    ex.printStackTrace();
}
Gathers all named components in a container.
@param c the component
@param namedComponents a map into which to enter the component names and components
*/
private static void getComponentBindings(Component c, Map<String, Component> namedComponents)
{
    String name = c.getName();
    if (name != null) { namedComponents.put(name, c); }
    if (c instanceof Container)
    {
        for (Component child : ((Container) c).getComponents())
            getComponentBindings(child, namedComponents);
    }
}

/**
 * Adds a listener to an object whose listener method executes a script.
 * @param beanName the name of the bean to which the listener should be added
 * @param eventName the name of the listener type, such as "action" or "change"
 * @param scriptCode the script code to be executed
 * @param engine the engine that executes the code
 * @param bindings the bindings for the execution
 * @throws IntrospectionException
 */
private static void addListener(String beanName, String eventName, final String scriptCode,
    final ScriptEngine engine, Map<String, Component> components)
    throws ReflectiveOperationException, IntrospectionException
{
    Object bean = components.get(beanName);
    EventSetDescriptor descriptor = getEventSetDescriptor(bean, eventName);
    if (descriptor == null) return;
    descriptor.addListenerMethod().invoke(bean,
        Proxy.newProxyInstance(null, new Class[]{ descriptor.getListenerType() },
            (proxy, method, args) ->
            {
                engine.eval(scriptCode);
                return null;
            }));
}

private static EventSetDescriptor getEventSetDescriptor(Object bean, String eventName)
    throws IntrospectionException
{
    for (EventSetDescriptor descriptor : Introspector.getBeanInfo(bean.getClass())
        .getEventSetDescriptors())
    { (Continues)
Listing 8.1 (Continued)

```java
if (descriptor.getName().equals(eventName)) return descriptor;
return null;
```

Listing 8.2  buttons1/ButtonFrame.java

```java
package buttons1;
import javax.swing.*;
/**
 * A frame with a button panel.
 * @version 1.00 2007-11-02
 * @author Cay Horstmann
 */
public class ButtonFrame extends JFrame {
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 200;
    private JPanel panel;
    private JButton yellowButton;
    private JButton blueButton;
    private JButton redButton;
    public ButtonFrame()
    {
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
        panel = new JPanel();
        panel.setName("panel");
        add(panel);
        yellowButton = new JButton("Yellow");
        yellowButton.setName("yellowButton");
        blueButton = new JButton("Blue");
        blueButton.setName("blueButton");
        redButton = new JButton("Red");
        redButton.setName("redButton");
        panel.add(yellowButton);
        panel.add(blueButton);
        panel.add(redButton);
    }
}
```
8.2 The Compiler API

In the preceding sections, you saw how to interact with code in a scripting language. Now we turn to a different scenario: Java programs that compile Java code. There are quite a few tools that need to invoke the Java compiler, such as:

- Development environments
- Java teaching and tutoring programs
- Build and test automation tools
- Templating tools that process snippets of Java code, such as JavaServer Pages (JSP)

In the past, applications invoked the Java compiler by calling undocumented classes in the `jdk/lib/tools.jar` library. Nowadays, a public API for compilation is a part of the Java platform, and it is no longer necessary to use `tools.jar`. This section explains the compiler API.

8.2.1 Compiling the Easy Way

It is very easy to invoke the compiler. Here is a sample call:

```java
JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();
OutputStream outStream = . . .;
OutputStream errStream = . . .;
int result = compiler.run(null, outStream, errStream, "-sourcepath", "src", "Test.java");
```

A result value of 0 indicates successful compilation.

The compiler sends output and error messages to the provided streams. You can set these parameters to `null`, in which case `System.out` and `System.err` are used. The first parameter of the `run` method is an input stream. As the compiler takes no console input, you can always leave it as `null`. (The `run` method is inherited from a generic `Tool` interface, which allows for tools that read input.)

The remaining parameters of the `run` method are simply the arguments that you would pass to `javac` if you invoked it on the command line. These can be options or file names.

8.2.2 Using Compilation Tasks

You can have even more control over the compilation process with a `CompilationTask` object. In particular, you can

- Control the source of program code—for example, by providing code in a string builder instead of a file
• Control the placement of class files—for example, by storing them in a database
• Listen to error and warning messages as they occur during compilation
• Run the compiler in the background

The location of source and class files is controlled by a `JavaFileManager`. It is responsible for determining `JavaFileObject` instances for source and class files. A `JavaFileObject` can correspond to a disk file, or it can provide another mechanism for reading and writing its contents.

To listen to error messages, install a `DiagnosticListener`. The listener receives a `Diagnostic` object whenever the compiler reports a warning or error message. The `DiagnosticCollector` class implements this interface. It simply collects all diagnostics so that you can iterate through them after the compilation is complete.

A `Diagnostic` object contains information about the problem location (including file name, line number, and column number) as well as a human-readable description.

To obtain a `CompilationTask` object, call the `getTask` method of the `JavaCompiler` class. You need to specify:

- A `Writer` for any compiler output that is not reported as a `Diagnostic`, or `null` to use `System.err`
- A `JavaFileManager`, or `null` to use the compiler’s standard file manager
- A `DiagnosticListener`
- Option strings, or `null` for no options
- Class names for annotation processing, or `null` if none are specified (we’ll discuss annotation processing later in this chapter)
- `JavaFileObject` instances for source files

You need to provide the last three arguments as `Iterable` objects. For example, a sequence of options might be specified as

```java
Iterable<String> options = Arrays.asList("-g", "-d", "classes");
```

Alternatively, you can use any collection class.

If you want the compiler to read source files from disk, you can ask the `StandardJavaFileManager` to translate the file name strings or `File` objects to `JavaFileObject` instances. For example,

```java
StandardJavaFileManager fileManager = compiler.getStandardFileManager(null, null, null);
Iterable<JavaFileObject> fileObjects = fileManager.getJavaFileObjectsFromStrings(fileNames);
```

However, if you want the compiler to read source code from somewhere other than a disk file, you need to supply your own `JavaFileObject` subclass. Listing 8.3
shows the code for a source file object with data contained in a StringBuilder. The class extends the SimpleJavaFileObject convenience class and overrides the getCharContent method to return the content of the string builder. We’ll use this class in our example program in which we dynamically produce the code for a Java class and then compile it.

The CompilationTask interface extends the Callable<Boolean> interface. You can pass it to an Executor for execution in another thread, or you can simply invoke the call method. A return value of Boolean.FALSE indicates failure.

```
Callable<Boolean> task = new JavaCompiler.CompilationTask(null, fileManager, diagnostics, options, null, fileObjects);
if (!task.call())
    System.out.println("Compilation failed");
```

If you simply want the compiler to produce class files on disk, you need not customize the JavaFileManager. However, our sample application will generate class files in byte arrays and later read them from memory, using a special class loader. Listing 8.4 defines a class that implements the JavaFileObject interface. Its openOutputStream method returns the ByteArrayOutputStream into which the compiler will deposit the bytecodes.

It turns out a bit tricky to tell the compiler’s file manager to use these file objects. The library doesn’t supply a class that implements the StandardJavaFileManager interface. Instead, you subclass the ForwardingJavaFileManager class that delegates all calls to a given file manager. In our situation, we only want to change the getJavaFileForOutput method. We achieve this with the following outline:

```
JavaFileManager fileManager = compiler.getStandardFileManager(diagnostics, null, null);
fileManager = new ForwardingJavaFileManager<JavaFileManager>(fileManager)
{
    public JavaFileObject getJavaFileForOutput(Location location, final String className,
        Kind kind, FileObject sibling) throws IOException
    {
        return custom file object
    }
};
```

In summary, call the run method of the JavaCompiler task if you simply want to invoke the compiler in the usual way, reading and writing disk files. You can capture the output and error messages, but you need to parse them yourself.

If you want more control over file handling or error reporting, use the CompilationTask interface instead. Its API is quite complex, but you can control every aspect of the compilation process.
```java
Listing 8.3 compiler/StringBuilderJavaSource.java

```package compiler;```
```java```
```java
import java.net.*;
import javax.tools.*;

/**
 * A Java source that holds the code in a string builder.
 * @version 1.00 2007-11-02
 * @author Cay Horstmann
 */
public class StringBuilderJavaSource extends SimpleJavaFileObject
{
    private StringBuilder code;

    /**
     * Constructs a new StringBuilderJavaSource.
     * @param name the name of the source file represented by this file object
     */
    public StringBuilderJavaSource(String name)
    {
        super(URI.create("string://" + name.replace('.', '/') + Kind.SOURCE.extension),
            Kind.SOURCE);
        code = new StringBuilder();
    }

    public CharSequence getCharContent(boolean ignoreEncodingErrors)
    {
        return code;
    }

    public void append(String str)
    {
        code.append(str);
        code.append("\n");
    }
}

```

```java
Listing 8.4 compiler/ByteArrayJavaClass.java

```package compiler;```
```java```
```java
import java.io.*;
import java.net.*;
import javax.tools.*;

```
/**
 * A Java class that holds the bytecodes in a byte array.
 * @version 1.00 2007-11-02
 * @author Cay Horstmann
 */
public class ByteArrayJavaClass extends SimpleJavaFileObject {
  private ByteArrayOutputStream stream;

  /**
   * Constructs a new ByteArrayJavaClass.
   * @param name the name of the class file represented by this file object
   */
  public ByteArrayJavaClass(String name) {
    super(URI.create("bytes:///" + name), Kind.CLASS);
    stream = new ByteArrayOutputStream();
  }

  public OutputStream openOutputStream() throws IOException {
    return stream;
  }

  public byte[] getBytes() {
    return stream.toByteArray();
  }
}

javatools.Tool

- int run(InputStream in, OutputStream out, OutputStream err, String... arguments)
  runs the tool with the given input, output, and error streams and the given
  arguments. Returns 0 for success, a nonzero value for failure.

javatools.JavaCompiler

- StandardJavaFileManager getStandardFileManager(DiagnosticListener<? super JavaFileObject>
  diagnosticListener, Locale locale, Charset charset)
  gets the standard file manager for this compiler. You can supply null for default
  error reporting, locale, and character set.

(Continues)
### `javax.tools.JavaCompiler` 6 (Continued)

- `JavaCompiler.CompilationTask getTask(Writer out, JavaFileManager fileManager, DiagnosticListener<? super JavaFileObject> diagnosticListener, Iterable<String> options, Iterable<String> classesForAnnotationProcessing, Iterable<? extends JavaFileObject> sourceFiles)`

  gets a compilation task that, when called, will compile the given source files. See the discussion in the preceding section for details.

### `javax.tools.StandardJavaFileManager` 6

- `Iterable<? extends JavaFileObject> getJavaFileObjectsFromStrings(Iterable<String> fileNames)`
- `Iterable<? extends JavaFileObject> getJavaFileObjectsFromFiles(Iterable<? extends File> files)`

  translates a sequence of file names or files into a sequence of `JavaFileObject` instances.

### `javax.tools.JavaCompiler.CompilationTask` 6

- `Boolean call()`

  performs the compilation task.

### `javax.tools.DiagnosticCollector<S>` 6

- `DiagnosticCollector()`

  constructs an empty collector.
- `List<Diagnostic<? extends S>> getDiagnostics()`

  gets the collected diagnostics.

### `javax.tools.Diagnostic<S>` 6

- `S getSource()`

  gets the source object associated with this diagnostic.
- `Diagnostic.Kind getKind()`

  gets the type of this diagnostic—one of `ERROR`, `WARNING`, `MANDATORY_WARNING`, `NOTE`, or `OTHER`.  

(Continues)
Strings setMessage(Locale locale) gets the message describing the issue raised in this diagnostic. Pass null for the default locale.

• long getLineNumber()
• long getColumnNumber()
gets the position of the issue raised in this diagnostic.

CharSequence getCharContent(boolean ignoreEncodingErrors)
Override this method for a file object that represents a source file and produces the source code.

OutputStream openOutputStream()
Override this method for a file object that represents a class file and produces a stream to which the bytecodes can be written.

protected ForwardingJavaFileManager(M fileManager)
constructs a JavaFileManager that delegates all calls to the given file manager.

FileObject getFileForOutput(JavaFileManager.Location location, String className, JavaFileObject.Kind kind, FileObject sibling)
intercept this call if you want to substitute a file object for writing class files; kind is one of SOURCE, CLASS, HTML, or OTHER.

8.2.3 An Example: Dynamic Java Code Generation

In the JSP technology for dynamic web pages, you can mix HTML with snippets of Java code, for example

<p>The current date and time is <b><%= new java.util.Date() %></b>.</p>

The JSP engine dynamically compiles the Java code into a servlet. In our sample application, we use a simpler example and generate dynamic Swing code instead. The idea is that you use a GUI builder to lay out the components in a frame and specify the behavior of the components in an external file. Listing 8.5 shows a
very simple example of a frame class, and Listing 8.6 shows the code for the button actions. Note that the constructor of the frame class calls an abstract method addEventHandlers. Our code generator will produce a subclass that implements the addEventHandlers method, adding an action listener for each line in the action.properties file. (We leave it as the proverbial exercise to the reader to extend the code generation to other event types.)

We place the subclass into a package with the name \textit{x}, which we hope is not used anywhere else in the program. The generated code has the form

```java
package x;
public class Frame extends SuperclassName {
    protected void addEventHandlers()
    {
        componentName1.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent) {
                code for event handler
            }
        });
        // repeat for the other event handlers ...
    }
}
```

The \texttt{buildSource} method in the program of Listing 8.7 builds up this code and places it into a \texttt{StringBuilderJavaSource} object. That object is passed to the Java compiler.

We use a \texttt{ForwardingJavaFileManager} with a \texttt{getJavaFileForOutput} method that constructs a \texttt{ByteArrayJavaClass} object for every class in the \textit{x} package. These objects capture the class files generated when the \textit{x.Frame} class is compiled. The method adds each file object to a list before returning it so that we can locate the bytecodes later. Note that compiling the \textit{x.Frame} class produces a class file for the main class and one class file per listener class.

After compilation, we build a map that associates class names with bytecode arrays. A simple class loader (shown in Listing 8.8) loads the classes stored in this map.

We ask the class loader to load the class that we just compiled. Then, we construct and display the application’s frame class.

```java
ClassLoader loader = new MapClassLoader(byteCodeMap);
Class<?> cl = loader.loadClass("x.Frame");
Frame frame = (JFrame) cl.newInstance();
frame.setVisible(true);
```

When you click the buttons, the background color changes in the usual way. To see that the actions are dynamically compiled, change one of the lines in action.properties, for example, like this:
yellowButton = panel.setBackground(java.awt.Color.YELLOW); yellowButton.setEnabled(false);

Run the program again. Now the Yellow button is disabled after you click it. Also, have a look at the code directories. You will not find any source or class files for the classes in the x package. This example demonstrates how you can use dynamic compilation with in-memory source and class files.

Listing 8.5 buttons2/ButtonFrame.java

```java
package buttons2;
import javax.swing.*;

/**
 * A frame with a button panel.
 * @version 1.00 2007-11-02
 * @author Cay Horstmann
 */
public abstract class ButtonFrame extends JFrame {
    public static final int DEFAULT_WIDTH = 300;
    public static final int DEFAULT_HEIGHT = 200;

    protected JPanel panel;
    protected JButton yellowButton;
    protected JButton blueButton;
    protected JButton redButton;

    protected abstract void addEventHandlers();

    public ButtonFrame() {
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
        panel = new JPanel();
        add(panel);

        yellowButton = new JButton("Yellow");
        blueButton = new JButton("Blue");
        redButton = new JButton("Red");

        panel.add(yellowButton);
        panel.add(blueButton);
        panel.add(redButton);

        addEventHandlers();
    }
}
```

8.2 The Compiler API
Listing 8.6  buttons2/action.properties

1. yellowButton=panel.setBackground(java.awt.Color.YELLOW);
2. blueButton=panel.setBackground(java.awt.Color.BLUE);

Listing 8.7  compiler/CompilerTest.java

```
package compiler;

import java.awt.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.tools.*;
import javax.tools.JavaFileObject.*;

/**
* @version 1.01 2016-05-10
* @author Cay Horstmann
* /
public class CompilerTest
{
    public static void main(final String[] args) throws IOException, ClassNotFoundException
    {
        JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();

        final List<ByteArrayJavaClass> classFileObjects = new ArrayList<>();
        DiagnosticCollector<JavaFileObject> diagnostics = new DiagnosticCollector<>();

        JavaFileManager fileManager = compiler.getStandardFileManager(diagnostics, null, null);
        fileManager = new ForwardingJavaFileManager<JavaFileManager>(fileManager)
        {
            public JavaFileObject getJavaFileForOutput(Location location, final String className,
            Kind kind, FileObject sibling) throws IOException
            {
                if (className.startsWith("x."))
                {
                    ByteArrayJavaClass fileObject = new ByteArrayJavaClass(className);
                    classFileObjects.add(fileObject);
                    return fileObject;
                }
                else return super.getJavaFileForOutput(location, className, kind, sibling);
            }
        };
    }
```
String frameClassName = args.length == 0 ? "buttons2.ButtonFrame" : args[0];
JavaFileObject source = buildSource(frameClassName);
JavaCompiler.CompilationTask task = compiler.getTask(null, fileManager, diagnostics, null,
    null, Arrays.asList(source));
Boolean result = task.call();

for (Diagnostic<? extends JavaFileObject> d : diagnostics.getDiagnostics())
    System.out.println(d.getKind() + " : " + d.getMessage(null));
fileManager.close();
if (!result)
    { System.out.println("Compilation failed.");
        System.exit(1); }

EventQueue.invokeLater(() -> {
    try {
        Map<String, byte[]> byteCodeMap = new HashMap<>();
        for (ByteArrayJavaClass cl : classFileObjects)
            byteCodeMap.put(cl.getName().substring(1), cl.getBytes());
        ClassLoader loader = new MapClassLoader(byteCodeMap);
        JFrame frame = (JFrame) loader.loadClass("x.Frame").newInstance();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setTitle("CompilerTest");
        frame.setVisible(true);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
});

/* Builds the source for the subclass that implements the addEventHandlers method.
 * @return a file object containing the source in a string builder
 */
static JavaFileObject buildSource(String superclassName) throws IOException, ClassNotFoundException {
    StringBuilderJavaSource source = new StringBuilderJavaSource("x.Frame");
    source.append("package x;
    public class Frame extends " + superclassName + " {
        protected void addEventHandlers() {
            final Properties props = new Properties();
            
            // (Continues)
Listing 8.7  (Continued)

```java
props.load(Class.forName(superclassName).getResourceAsStream("action.properties"));
for (Map.Entry<Object, Object> e : props.entrySet())
{
    String beanName = (String) e.getKey();
    String eventCode = (String) e.getValue();
    source.append(beanName + ".addActionListener(event -> {");
    source.append(eventCode);
    source.append("} );");
}
source.append("} }");
return source;
```

Listing 8.8  compiler/MapClassLoader.java

```java
package compiler;

import java.util.*;

/**
 * A class loader that loads classes from a map whose keys are class names and whose values are
 * byte code arrays.
 * @version 1.00 2007-11-02
 * @author Cay Horstmann
 */
public class MapClassLoader extends ClassLoader
{
    private Map<String, byte[]> classes;

    public MapClassLoader(Map<String, byte[]> classes)
    {
        this.classes = classes;
    }

    protected Class<?> findClass(String name) throws ClassNotFoundException
    {
        byte[] classBytes = classes.get(name);
        if (classBytes == null) throw new ClassNotFoundException(name);
        Class<?> c1 = defineClass(name, classBytes, 0, classBytes.length);
        if (c1 == null) throw new ClassNotFoundException(name);
        return c1;
    }
}
```
8.3 Using Annotations

Annotations are tags that you insert into your source code so that some tool can process them. The tools can operate on the source level, or they can process class files into which the compiler has placed annotations.

Annotations do not change the way in which your programs are compiled. The Java compiler generates the same virtual machine instructions with or without the annotations.

To benefit from annotations, you need to select a processing tool. Use annotations that your processing tool understands, then apply the processing tool to your code.

There is a wide range of uses for annotations, and that generality can be confusing at first. Here are some uses for annotations:

- Automatic generation of auxiliary files, such as deployment descriptors or bean information classes
- Automatic generation of code for testing, logging, transaction semantics, and so on

8.3.1 An Introduction into Annotations

We’ll start our discussion of annotations with the basic concepts and put them to use in a concrete example: We will mark methods as event listeners for AWT components, and show you an annotation processor that analyzes the annotations and hooks up the listeners. We’ll then discuss the syntax rules in detail and finish the chapter with two advanced examples of annotation processing. One of them processes source-level annotations, the other uses the Apache Bytecode Engineering Library to process class files, injecting additional bytecodes into annotated methods.

Here is an example of a simple annotation:

```java
public class MyClass {
    . . .
    @Test public void checkRandomInsertions()
}
```

The annotation `@Test` annotates the `checkRandomInsertions` method.

In Java, an annotation is used like a modifier and is placed before the annotated item without a semicolon. (A modifier is a keyword such as `public` or `static`.) The name of each annotation is preceded by an `@` symbol, similar to Javadoc comments.
However, Javadoc comments occur inside /** */ delimiters, whereas annotations are part of the code.

By itself, the @Test annotation does not do anything. It needs a tool to be useful. For example, the JUnit 4 testing tool (available at http://junit.org) calls all methods that are labeled @Test when testing a class. Another tool might remove all test methods from a class file so they are not shipped with the program after it has been tested.

Annotations can be defined to have elements, such as

@Test(timeout="10000")

These elements can be processed by the tools that read the annotations. Other forms of elements are possible; we’ll discuss them later in this chapter.

Besides methods, you can annotate classes, fields, and local variables—an annotation can be anywhere you could put a modifier such as public or static. In addition, as you will see in Section 8.4, “Annotation Syntax,” on p. 462, you can annotate packages, parameter variables, type parameters, and type uses.

Each annotation must be defined by an annotation interface. The methods of the interface correspond to the elements of the annotation. For example, the JUnit Test annotation is defined by the following interface:

```java
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Test {
  long timeout() default 0L;
  ...
}
```

The @interface declaration creates an actual Java interface. Tools that process annotations receive objects that implement the annotation interface. A tool would call the timeout method to retrieve the timeout element of a particular Test annotation.

The Target and Retention annotations are meta-annotations. They annotate the Test annotation, marking it as an annotation that can be applied to methods only and is retained when the class file is loaded into the virtual machine. We’ll discuss these in detail in Section 8.5.3, “Meta-Annotations,” on p. 472.

You have now seen the basic concepts of program metadata and annotations. In the next section, we’ll walk through a concrete example of annotation processing.
8.3.2 An Example: Annotating Event Handlers

One of the more boring tasks in user interface programming is the wiring of listeners to event sources. Many listeners are of the form

```java
myButton.addActionListener(() -> doSomething());
```

In this section, we’ll design an annotation to reverse the wiring. The annotation, defined in Listing 8.11, is used as follows:

```java
@ActionListenerFor(source="myButton") void doSomething() { ... }
```

The programmer no longer has to make calls to `addActionListener`. Instead, each method is tagged with an annotation. Listing 8.10 shows the `ButtonFrame` class from Volume I, Chapter 11, reimplemented with these annotations.

We also need to define an annotation interface. The code is in Listing 8.11.

Of course, the annotations don’t do anything by themselves. They sit in the source file. The compiler places them in the class file, and the virtual machine loads them. We now need a mechanism to analyze them and install action listeners. That is the job of the `ActionListenerInstaller` class. The `ButtonFrame` constructor calls

```java
ActionListenerInstaller.processAnnotations(this);
```

The static `processAnnotations` method enumerates all methods of the object it received. For each method, it gets the `ActionListenerFor` annotation object and processes it.

```java
Class<?> cl = obj.getClass();
for (Method m : cl.getDeclaredMethods())
{
    ActionListenerFor a = m.getAnnotation(ActionListenerFor.class);
    if (a != null) ... 
}
```

Here, we use the `getAnnotation` method defined in the `AnnotatedElement` interface. The classes `Method`, `Constructor`, `Field`, `Class`, and `Package` implement this interface.

The name of the source field is stored in the annotation object. We retrieve it by calling the `source` method, and then look up the matching field.

```java
String fieldName = a.source();
Field f = cl.getDeclaredField(fieldName);
```

This shows a limitation of our annotation. The source element must be the name of a field. It cannot be a local variable.
The remainder of the code is rather technical. For each annotated method, we construct a proxy object, implementing the ActionListener interface, with an actionPerformed method that calls the annotated method. (For more information about proxies, see Volume I, Chapter 6.) The details are not important. The key observation is that the functionality of the annotations was established by the processAnnotations method.

Figure 8.1 shows how annotations are handled in this example.

![Figure 8.1 Processing annotations at runtime](image)

In this example, the annotations were processed at runtime. It is also possible to process them at the source level: A source code generator would then produce the code for adding the listeners. Alternatively, the annotations can be processed at the bytecode level: A bytecode editor could inject the calls to addActionListener into the frame constructor. This sounds complex, but libraries are available to make this task relatively straightforward. You can see an example in Section 8.7, “Bytecode Engineering,” on p. 481.

Our example was not intended as a serious tool for user interface programmers. A utility method for adding a listener could be just as convenient for the programmer as the annotation. (In fact, the java.beans.EventHandler class tries to do just that. You could make the class truly useful by supplying a method that adds the event handler instead of just constructing it.)

However, this example shows the mechanics of annotating a program and of analyzing the annotations. Having seen a concrete example, you are now more prepared (we hope) for the following sections that describe the annotation syntax in complete detail.
package runtimeAnnotations;

import java.awt.event.*;
import java.lang.reflect.*;

/**
 * @version 1.00 2004-08-17
 * @author Cay Horstmann
 */
public class ActionListenerInstaller
{
    /**
     * Processes all ActionListenerFor annotations in the given object.
     * @param obj an object whose methods may have ActionListenerFor annotations
     */
    public static void processAnnotations(Object obj)
    {
        try
        {
            Class<?> cl = obj.getClass();
            for (Method m : cl.getDeclaredMethods())
            {
                ActionListenerFor a = m.getAnnotation(ActionListenerFor.class);
                if (a != null)
                {
                    Field f = cl.getDeclaredField(a.source());
                    f.setAccessible(true);
                    addListener(f.get(obj), obj, m);
                }
            }
        }
        catch (ReflectiveOperationException e)
        {
            e.printStackTrace();
        }
    }

    /**
     * Adds an action listener that calls a given method.
     * @param source the event source to which an action listener is added
     * @param param the implicit parameter of the method that the listener calls
     * @param m the method that the listener calls
     */
    public static void addListener(Object source, final Object param, final Method m)
    throws ReflectiveOperationException
    {
        (Continues)
Listing 8.9  (Continued)

```java
InvocationHandler handler = new InvocationHandler()
{
    public Object invoke(Object proxy, Method mm, Object[] args) throws Throwable
    {
        return m.invoke(param);
    }
};

Object listener = Proxy.newProxyInstance(null,
    new Class[] { java.awt.event.ActionListener.class }, handler);
Method adder = source.getClass().getMethod("addActionListener", ActionListener.class);
adder.invoke(source, listener);
```

Listing 8.10  buttons3/ButtonFrame.java

```java
package buttons3;

import java.awt.*;
import javax.swing.*;
import runtimeAnnotations.*;

/**
 * A frame with a button panel.
 * @version 1.00 2004-08-17
 * @author Cay Horstmann
 */
public class ButtonFrame extends JFrame
{
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 200;
    private JPanel panel;
    private JButton yellowButton;
    private JButton blueButton;
    private JButton redButton;

    public ButtonFrame()
    {
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
        panel = new JPanel();
        add(panel);
    }
```
yellowButton = new JButton("Yellow");
blueButton = new JButton("Blue");
redButton = new JButton("Red");

panel.add(yellowButton);
panel.add(blueButton);
panel.add(redButton);

ActionListenerInstaller.processAnnotations(this);

@ActionListenerFor(source = "yellowButton")
public void yellowBackground() {
    panel.setBackground(Color.YELLOW);
}

@ActionListenerFor(source = "blueButton")
public void blueBackground() {
    panel.setBackground(Color.BLUE);
}

@ActionListenerFor(source = "redButton")
public void redBackground() {
    panel.setBackground(Color.RED);
}
8.4 Annotation Syntax

In the following sections, we cover everything you need to know about the annotation syntax.

8.4.1 Annotation Interfaces

An annotation is defined by an annotation interface:

```java
modifiers @interface AnnotationName
{
    elementDeclaration1
    elementDeclaration2
    ...
}
```

Each element declaration has the form

```java
type elementName();
```

or

```java
type elementName() default value;
```

For example, the following annotation has two elements, assignedTo and severity:
public @interface BugReport
{
    String assignedTo() default "[none]";
    int severity();
}

All annotation interfaces implicitly extend the java.lang.annotation.Annotation interface. That interface is a regular interface, not an annotation interface. See the API notes at the end of this section for the methods provided by this interface. You cannot extend annotation interfaces. In other words, all annotation interfaces directly extend java.lang.annotation.Annotation. You never supply classes that implement annotation interfaces.

The methods of an annotation interface have no parameters and no throws clauses. They cannot be default or static methods, and they cannot have type parameters.

The type of an annotation element is one of the following:

- A primitive type (int, short, long, byte, char, double, float, or boolean)
- String
- Class (with an optional type parameter such as Class<? extends MyClass>)
- An enum type
- An annotation type
- An array type

Here are examples of valid element declarations:

```java
public @interface BugReport
{
    enum Status { UNCONFIRMED, CONFIRMED, FIXED, NOTABUG };
    boolean showStopper() default false;
    String assignedTo() default "[none]";
    Class<?> testCase() default Void.class;
    Status status() default Status.UNCONFIRMED;
    Reference ref() default @Reference(); // an annotation type
    String[] reportedBy();
}
```

```java
java.lang.annotation.Annotation 5.0
```

- Class<? extends Annotation> annotationType()

returns the Class object that represents the annotation interface of this annotation object. Note that calling getClass on an annotation object would return the actual class, not the interface.

(Continues)
java.lang.annotation.Annotation 5.0 (Continued)

- boolean equals(Object other)
  returns true if other is an object that implements the same annotation interface as this annotation object and if all elements of this object and other are equal.

- int hashCode()
  returns a hash code, compatible with the equals method, derived from the name of the annotation interface and the element values.

- String toString()
  returns a string representation that contains the annotation interface name and the element values; for example, @BugReport(assignedTo=[none], severity=0).

8.4.2 Annotations

Each annotation has the format

@AnnotationName(elementName1=value1, elementName2=value2, ...)

For example,

@BugReport(assignedTo="Harry", severity=10)

The order of the elements does not matter. The annotation

@BugReport(severity=10, assignedTo="Harry")

is identical to the preceding one.

The default value of the declaration is used if an element value is not specified. For example, consider the annotation

@BugReport(severity=10)

The value of the assignedTo element is the string "[none]."

**CAUTION:** Defaults are not stored with the annotation; instead, they are dynamically computed. For example, if you change the default for the assignedTo element to "[]" and recompile the BugReport interface, the annotation @BugReport(severity=10) will use the new default, even in class files that have been compiled before the default changed.

Two special shortcuts can simplify annotations.
If no elements are specified, either because the annotation doesn’t have any or because all of them use the default value, you don’t need to use parentheses. For example,

```java
@BugReport
```

is the same as

```java
@BugReport(assignedTo="[none]", severity=0)
```

Such an annotation is called a *marker annotation*.

The other shortcut is the *single value annotation*. If an element has the special name `value` and no other element is specified, you can omit the element name and the `=` symbol. For example, had we defined the `ActionListenerFor` annotation interface of the preceding section as

```java
public @interface ActionListenerFor {
    String value();
}
```

then the annotations could be written as

```java
@ActionListenerFor("yellowButton")
```

instead of

```java
@ActionListenerFor(value="yellowButton")
```

An item can have multiple annotations:

```java
@Test
@BugReport(showStopper=true, reportedBy="Joe")
public void checkRandomInsertions()
```

If the author of an annotation declared it to be repeatable, you can repeat the same annotation multiple times:

```java
@BugReport(showStopper=true, reportedBy="Joe")
@BugReport(reportedBy={"Harry", "Carl"})
public void checkRandomInsertions()
```

**NOTE:** Since annotations are evaluated by the compiler, all element values must be compile-time constants. For example,

```java
@BugReport(showStopper=true, assignedTo="Harry", testCase=MyTestCase.class,
    status=BugReport.Status.CONFIRMED, . . .)
```
CAUTION: An annotation element can never be set to null. Not even a default of null is permissible. This can be rather inconvenient in practice. You will need to find other defaults, such as "" or Void.class.

If an element value is an array, enclose its values in braces:

```java
@BugReport(. . ., reportedBy={"Harry", "Carl"})
```

You can omit the braces if the element has a single value:

```java
@BugReport(. . ., reportedBy="Joe") // OK, same as {"Joe"}
```

Since an annotation element can be another annotation, you can build arbitrarily complex annotations. For example,

```java
@BugReport(ref=@Reference(id="3352627"), . . .)
```

NOTE: It is an error to introduce circular dependencies in annotations. For example, BugReport has an element of the annotation type Reference, therefore Reference cannot have an element of type BugReport.

8.4.3 Annotating Declarations

There are many places where annotations can occur. They fall into two categories: declarations and type uses. Declaration annotations can appear at the declarations of

- Packages
- Classes (including enum)
- Interfaces (including annotation interfaces)
- Methods
- Constructors
- Instance fields (including enum constants)
- Local variables
- Parameter variables
- Type parameters

For classes and interfaces, put the annotations before the class or interface keyword:

```java
@Entity public class User { . . . }
```

For variables, put them before the type:

```java
@SuppressWarnings("unchecked") List<User> users = . . .;
public User getUser(@Param("id") String userId)
```
A type parameter in a generic class or method can be annotated like this:

```java
public class Cache<@Immutable V> { . . . }
```

A package is annotated in a file `package-info.java` that contains only the package statement preceded by annotations.

```java
/**
 * Package-level Javadoc
 */
@GPL(version="3")
package com.horstmann.corejava;
import org.gnu.GPL;
```

**NOTE:** Annotations for local variables can only be processed at the source level. Class files do not describe local variables. Therefore, all local variable annotations are discarded when a class is compiled. Similarly, annotations for packages are not retained beyond the source level.

### 8.4.4 Annotating Type Uses

A declaration annotation provides some information about the item being declared. For example, in the declaration

```java
public User getUser(@NonNull String userId)
```

it is asserted that the `userId` parameter is not null.

**NOTE:** The `@NonNull` annotation is a part of the Checker Framework (http://types.cs.washington.edu/checker-framework). With that framework, you can include assertions in your program, such that a parameter is non-null or that a `String` contains a regular expression. A static analysis tool then checks whether the assertions are valid in a given body of source code.

Now suppose we have a parameter of type `List<String>`, and we want to express that all of the strings are non-null. That is where type use annotations come in. Place the annotation before the type argument: `List<@NonNull String>`. Type use annotations can appear in the following places:

- With generic type arguments: `List<@NonNull String>`, `Comparator.<@NonNull String> reverseOrder()`.
- In any position of an array: `@NonNull String[][] words (words[i][j] is not null)`, `String @NonNull [][] words (words is not null)`, `String[] @NonNull [] words (words[i] is not null)`. 
• With superclasses and implemented interfaces: class Warning extends @LocalizedMessage Message.
• With constructor invocations: new @LocalizedMessage String(...).
• With casts and instanceof checks: (@LocalizedMessage String) text, if (text instanceof @LocalizedMessage String). (The annotations are only for use by external tools. They have no effect on the behavior of a cast or an instanceof check.)
• With exception specifications: public String read() throws @LocalizedMessage IOException.
• With wildcards and type bounds: List<? extends @LocalizedMessage Message>, List<? extends @LocalizedMessage Message>.
• With method and constructor references: @LocalizedMessage Message::getText.

There are a few type positions that cannot be annotated:

```java
@NonNull String.class  // ERROR: Cannot annotate class literal
import java.lang.@NonNull String; // ERROR: Cannot annotate import
```

You can place annotations before or after other modifiers such as private and static. It is customary (but not required) to put type use annotations after other modifiers, and declaration annotations before other modifiers. For example,

```java
private @NonNull String text; // Annotates the type use
@Id private String userId;  // Annotates the variable
```

**NOTE:** An annotation author needs to specify where a particular annotation can appear. If an annotation is permissible both for a variable and a type use, and it is used in a variable declaration, then both the variable and the type use are annotated. For example, consider

```java
public User getUser(@NonNull String userId)
```

If @NonNull can apply both to parameters and to type uses, the userId parameter is annotated, and the parameter type is @NonNull String.

### 8.4.5 Annotating this

Suppose you want to annotate parameters that are not being mutated by a method.

```java
public class Point
{
    public boolean equals(@ReadOnly Object other) {...}
}
```

Then a tool that processes this annotation would, upon seeing a call

```java
p.equals(q)
```
reason that \( q \) has not been changed.

But what about \( p \)?

When the method is called, the \( \texttt{this} \) variable is bound to \( p \). But \( \texttt{this} \) is never declared, so you cannot annotate it.

Actually, you can declare it, with a rarely used syntax variant, just so that you can add an annotation:

```java
public class Point
{
    public boolean equals(@ReadOnly Point this, @ReadOnly Object other) { ... }
}
```

The first parameter is called the receiver parameter. It must be named \( \texttt{this} \). Its type is the class that is being constructed.

---

**NOTE:** You can provide a receiver parameter only for methods, not for constructors. Conceptually, the \( \texttt{this} \) reference in a constructor is not an object of the given type until the constructor has completed. Instead, an annotation placed on the constructor describes a property of the constructed object.

A different hidden parameter is passed to the constructor of an inner class, namely the reference to the enclosing class object. You can make that parameter explicit as well:

```java
public class Sequence
{
    private int from;
    private int to;

    class Iterator implements java.util.Iterator<Integer>
    {
        private int current;

        public Iterator(@ReadOnly Sequence Sequence.this)
        {
            this.current = Sequence.this.from;
        }

        ... 
    }

    ... 
}
```

The parameter must be named just like when you refer to it, \( EnclosingClass\texttt{.this} \), and its type is the enclosing class.
8.5 Standard Annotations

Java SE defines a number of annotation interfaces in the java.lang, java.lang.annotation, and javax.annotation packages. Four of them are meta-annotations that describe the behavior of annotation interfaces. The others are regular annotations that you can use to annotate items in your source code. Table 8.2 shows these annotations. We’ll discuss them in detail in the following two sections.

Table 8.2 The Standard Annotations

<table>
<thead>
<tr>
<th>Annotation Interface</th>
<th>Applicable To</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deprecated</td>
<td>All</td>
<td>Marks item as deprecated.</td>
</tr>
<tr>
<td>SuppressWarnings</td>
<td>All but packages and annotations</td>
<td>Suppresses warnings of the given type.</td>
</tr>
<tr>
<td>SafeVarargs</td>
<td>Methods and constructors</td>
<td>Asserts that the varargs parameter is safe to use.</td>
</tr>
<tr>
<td>Override</td>
<td>Methods</td>
<td>Checks that this method overrides a superclass method.</td>
</tr>
<tr>
<td>FunctionalInterface</td>
<td>Interfaces</td>
<td>Marks an interface as functional (with a single abstract method).</td>
</tr>
<tr>
<td>PostConstruct</td>
<td>Methods</td>
<td>The marked method should be invoked immediately after construction or before removal.</td>
</tr>
<tr>
<td>PreDestroy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource</td>
<td>Classes, interfaces, methods, fields</td>
<td>On a class or interface, marks it as a resource to be used elsewhere. On a method or field, marks it for “injection.”</td>
</tr>
<tr>
<td>Resources</td>
<td>Classes, interfaces</td>
<td>Specifies an array of resources.</td>
</tr>
<tr>
<td>Generated</td>
<td>All</td>
<td>Marks an item as source code that has been generated by a tool.</td>
</tr>
<tr>
<td>Target</td>
<td>Annotations</td>
<td>Specifies the items to which this annotation can be applied.</td>
</tr>
<tr>
<td>Retention</td>
<td>Annotations</td>
<td>Specifies how long this annotation is retained.</td>
</tr>
<tr>
<td>Documented</td>
<td>Annotations</td>
<td>Specifies that this annotation should be included in the documentation of annotated items.</td>
</tr>
</tbody>
</table>

(Continues)
Table 8.2 (Continued)

<table>
<thead>
<tr>
<th>Annotation Interface</th>
<th>Applicable To</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited</td>
<td>Annotations</td>
<td>Specifies that this annotation, when applied to a class, is automatically inherited by its subclasses.</td>
</tr>
<tr>
<td>Repeatable</td>
<td>Annotations</td>
<td>Specifies that this annotation can be applied multiple times to the same item.</td>
</tr>
</tbody>
</table>

8.5.1 Annotations for Compilation

The @Deprecated annotation can be attached to any items whose use is no longer encouraged. The compiler will warn when you use a deprecated item. This annotation has the same role as the @deprecated Javadoc tag.

The @SuppressWarnings annotation tells the compiler to suppress warnings of a particular type, for example,

```java
@SuppressWarnings("unchecked")
```

The @Override annotation applies only to methods. The compiler checks that a method with this annotation really overrides a method from the superclass. For example, if you declare

```java
public MyClass
{
    @Override public boolean equals(MyClass other);
    ...
}
```

then the compiler will report an error. After all, the equals method does not override the equals method of the Object class because that method has a parameter of type Object, not MyClass.

The @Generated annotation is intended for use by code generator tools. Any generated source code can be annotated to differentiate it from programmer-provided code. For example, a code editor can hide the generated code, or a code generator can remove older versions of generated code. Each annotation must contain a unique identifier for the code generator. A date string (in ISO 8601 format) and a comment string are optional. For example,

```java
@Generated("com.horstmann.beanproperty", "2008-01-04T12:08:56.235-0700");
```
8.5.2 Annotations for Managing Resources

The @PostConstruct and @PreDestroy annotations are used in environments that control the lifecycle of objects, such as web containers and application servers. Methods tagged with these annotations should be invoked immediately after an object has been constructed or immediately before it is being removed.

The @Resource annotation is intended for resource injection. For example, consider a web application that accesses a database. Of course, the database access information should not be hardwired into the web application. Instead, the web container has some user interface for setting connection parameters and a JNDI name for a data source. In the web application, you can reference the data source like this:

```java
@Resource(name="jdbc/mydb")
private DataSource source;
```

When an object containing this field is constructed, the container “injects” a reference to the data source.

8.5.3 Meta-Annotations

The @Target meta-annotation is applied to an annotation, restricting the items to which the annotation applies. For example,

```java
@Target({ElementType.TYPE, ElementType.METHOD})
public @interface BugReport
```

Table 8.3 shows all possible values. They belong to the enumerated type ElementType. You can specify any number of element types, enclosed in braces.

An annotation without an @Target restriction can be applied to any item. The compiler checks that you apply an annotation only to a permitted item. For example, if you apply @BugReport to a field, a compile-time error results.

The @Retention meta-annotation specifies how long an annotation is retained. You can specify at most one of the values in Table 8.4. The default is RetentionPolicy.CLASS.

In Listing 8.11 on p. 461, the @ActionListenerFor annotation was declared with RetentionPolicy.RUNTIME because we used reflection to process annotations. In the following two sections, you will see examples of processing annotations at the source and class file levels.

The @Documented meta-annotation gives a hint to documentation tools such as Javadoc. Documented annotations should be treated just like other modifiers, such as protected or static, for documentation purposes. The use of other annotations is not
### Table 8.3 Element Types for the @Target Annotation

<table>
<thead>
<tr>
<th>Element Type</th>
<th>Annotation Applies To</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANNOTATION_TYPE</td>
<td>Annotation type declarations</td>
</tr>
<tr>
<td>PACKAGE</td>
<td>Packages</td>
</tr>
<tr>
<td>TYPE</td>
<td>Classes (including enum) and interfaces (including annotation types)</td>
</tr>
<tr>
<td>METHOD</td>
<td>Methods</td>
</tr>
<tr>
<td>CONSTRUCTOR</td>
<td>Constructors</td>
</tr>
<tr>
<td>FIELD</td>
<td>Fields (including enum constants)</td>
</tr>
<tr>
<td>PARAMETER</td>
<td>Method or constructor parameters</td>
</tr>
<tr>
<td>LOCAL_VARIABLE</td>
<td>Local variables</td>
</tr>
<tr>
<td>TYPE_PARAMETER</td>
<td>Type parameters</td>
</tr>
<tr>
<td>TYPE_USE</td>
<td>Uses of a type</td>
</tr>
</tbody>
</table>

### Table 8.4 Retention Policies for the @Retention Annotation

<table>
<thead>
<tr>
<th>Retention Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCE</td>
<td>Annotations are not included in class files.</td>
</tr>
<tr>
<td>CLASS</td>
<td>Annotations are included in class files, but the virtual machine need not load them.</td>
</tr>
<tr>
<td>RUNTIME</td>
<td>Annotations are included in class files and loaded by the virtual machine. They are available through the reflection API.</td>
</tr>
</tbody>
</table>

included in the documentation. For example, suppose we declare `@ActionListenerFor` as a documented annotation:

```java
@Documented
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface ActionListenerFor
```

Now the documentation of each annotated method contains the annotation, as shown in Figure 8.2.

If an annotation is transient (such as `@BugReport`), you should probably not document its use.
NOTE: It is legal to apply an annotation to itself. For example, the @Documented annotation is itself annotated as @Documented. Therefore, the Javadoc documentation for annotations shows whether they are documented.

The @Inherited meta-annotation applies only to annotations for classes. When a class has an inherited annotation, then all of its subclasses automatically have the same annotation. This makes it easy to create annotations that work as marker interfaces, such as Serializable.

In fact, an annotation @Serializable would be more appropriate than the Serializable marker interface with no methods. A class is serializable because there is runtime support for reading and writing its fields, not because of any principles of object-oriented design. An annotation describes this fact better than does interface inheritance. Of course, the Serializable interface was created in JDK 1.1, long before annotations existed.

Suppose you define an inherited annotation @Persistable to indicate that objects of a class can be saved in a database. Then the subclasses of persistent classes are automatically annotated as persistent.
When the persistence mechanism searches for objects to store in the database, it will detect both Employee and Manager objects.

As of Java SE 8, it is legal to apply the same annotation type multiple times to an item. For backward compatibility, the implementor of a repeatable annotation needs to provide a container annotation that holds the repeated annotations in an array.

Here is how to define the @TestCase annotation and its container:

```java
@Repeatable(TestCases.class)
@interface TestCase
{
    String params();
    String expected();
}

@interface TestCases
{
    TestCase[] value();
}
```

Whenever the user supplies two or more @TestCase annotations, they are automatically wrapped into a @TestCases annotation.

**CAUTION:** You have to be careful when processing repeatable annotations. If you call getAnnotation to look up a repeatable annotation, and the annotation was actually repeated, then you get null. That is because the repeated annotations were wrapped into the container annotation.

In that case, you should call getAnnotationsByType. That call “looks through” the container and gives you an array of the repeated annotations. If there was just one annotation, you get it in an array of length 1. With this method, you don’t have to worry about the container annotation.

### 8.6 Source-Level Annotation Processing

In the preceding section, you saw how to analyze annotations in a running program. Another use for annotation is the automatic processing of source files to produce more source code, configuration files, scripts, or whatever else one might want to generate.
8.6.1 Annotation Processors

Annotation processing is integrated into the Java compiler. During compilation, you can invoke annotation processors by running

```
javac -processor ProcessorClassName1,ProcessorClassName2,... sourceFiles
```

The compiler locates the annotations of the source files. Each annotation processor is executed in turn and given the annotations in which it expressed an interest. If an annotation processor creates a new source file, the process is repeated. Once a processing round yields no further source files, all source files are compiled.

**NOTE:** An annotation processor can only generate new source files. It cannot modify an existing source file.

An annotation processor implements the `Processor` interface, generally by extending the `AbstractProcessor` class. You need to specify which annotations your processor supports. In our case:

```java
@SupportedAnnotationTypes("com.horstmann.annotations.ToString")
@SupportedSourceVersion(SourceVersion.RELEASE_8)
public class ToStringAnnotationProcessor extends AbstractProcessor
{
    public boolean process(Set<? extends TypeElement> annotations,
                           RoundEnvironment currentRound)
    {
        ... 
    }
}
```

A processor can claim specific annotation types, wildcards such as "com.horstmann.*" (all annotations in the `com.horstmann` package or any subpackage), or even "*" (all annotations).

The `process` method is called once for each round, with the set of all annotations that were found in any files during this round, and a `RoundEnvironment` reference that contains information about the current processing round.

8.6.2 The Language Model API

Use the `language model` API for analyzing source-level annotations. Unlike the reflection API, which presents the virtual machine representation of classes and methods, the language model API lets you analyze a Java program according to the rules of the Java language.
The compiler produces a tree whose nodes are instances of classes that implement the `javax.lang.model.element.Element` interface and its subinterfaces: `TypeElement`, `VariableElement`, `ExecutableElement`, and so on. These are the compile-time analogs to the `Class`, `Field/Parameter`, `Method/Constructor` reflection classes.

I do not want to cover the API in detail, but here are the highlights that you need to know for processing annotations:

- The `RoundEnvironment` gives you a set of all elements annotated with a particular annotation, by calling the method
  ```java
  Set<? extends Element> getElementsAnnotatedWith(Class<? extends Annotation> a)
  ```
- The source-level equivalent of the `AnnotateElement` interface is `AnnotatedConstruct`. Use the methods
  ```java
  A getAnnotation(Class<A> annotationType)
  A[] getAnnotationsByType(Class<A> annotationType)
  ```
to get the annotation or repeated annotations for a given annotation class.
- A `TypeElement` represents a class or interface. The `getEnclosedElements` method yields a list of its fields and methods.
- Calling `getSimpleName` on an `Element` or `getQualifiedName` on a `TypeElement` yields a `Name` object that can be converted to a string with `toString`.

### 8.6.3 Using Annotations to Generate Source Code

As an example, we will use annotations to reduce the tedium of implementing `toString` methods. We can’t put these methods into the original classes—annotation processors can only produce new classes, not modify existing ones.

Therefore, we’ll add all methods into a utility class `ToStrings`:

```java
public class ToStrings {
    public static String toString(Point obj) {
        // Generated code
    }
    public static String toString(Rectangle obj) {
        // Generated code
    }
    // ...
}
```
public static String toString(Object obj)
{
    return Objects.toString(obj);
}

We don’t want to use reflection, so we annotate accessor methods, not fields:

@ToString
public class Rectangle
{
    ...
    @ToString(includeName=false) public Point getTopLeft() { return topLeft; }
    @ToString public int getWidth() { return width; }
    @ToString public int getHeight() { return height; }
}

The annotation processor should then generate the following source code:

public static String toString(Rectangle obj)
{
    StringBuilder result = new StringBuilder();
    result.append("Rectangle");
    result.append("[");
    result.append(toString(obj.getTopLeft()));
    result.append("",$");
    result.append(toString(obj.getWidth()));
    result.append("",$");
    result.append(toString(obj.getHeight()));
    result.append(""]");
    return result.toString();
}

The “boilerplate” code is in gray. Here is an outline of the method that produces the toString method for a class with given TypeElement:

private void writeToStringMethod(PrintWriter out, TypeElement te)
{
    String className = te.getQualifiedName().toString();  
    StringBuilder result = new StringBuilder();
    result.append("Rectangle");
    result.append("[");
    for (Element c : te.getEnclosedElements())
    {
        ann = c.getAnnotation(ToString.class);
        if (ann != null)
        {
            if (ann.includeName())
            {
                String methodName = ann.value();  
                result.append(methodName + ",");
            }
            else
            {
                String fieldName = c.getSimpleName();  
                result.append(toString(fieldValue));
                result.append("",$");
            }
        }
    }
    result.append("]");
    return result.toString();
}
if (ann.includeName()) Print code to add field name
Print code to append toString(obj.methodName())

Print code to return string

And here is an outline of the process method of the annotation processor. It creates a source file for the helper class and writes the class header and one method for each annotated class.

```java
public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment currentRound) {
    if (annotations.size() == 0) return true;
    try {
        JavaFileObject sourceFile = processingEnv.getFiler().createSourceFile(
            "com.horstmann.annotations.ToStrings");
        try (PrintWriter out = new PrintWriter(sourceFile.openWriter())) {
            Print code for package and class
            for (Element e : currentRound.getElementsAnnotatedWith(ToString.class)) {
                if (e instanceof TypeElement) {
                    TypeElement te = (TypeElement) e;
                    writeToStringMethod(out, te);
                }
            }
            Print code for toString(Object)
        }
    } catch (IOException ex) {
        processingEnv.getMessager().printMessage(
            Kind.ERROR, ex.getMessage());
    }
    return true;
}
```

For the tedious details, check the book’s companion code.

Note that the process method is called in subsequent rounds with an empty list of annotations. It then returns immediately so it doesn’t create the source file twice.

First compile the annotation processor, and then compile and run the test program as follows:
javac sourceAnnotations/ToStringAnnotationProcessor.java
javac -processor sourceAnnotations.ToStringAnnotationProcessor rect/*.java
java rect.SourceLevelAnnotationDemo

TIP: To see the rounds, run the javac command with the -XprintRounds flag:

Round 1:
input files: {rect.Point, rect.Rectangle, rect.SourceLevelAnnotationDemo}
annotations: [sourceAnnotations.ToString]
last round: false
Round 2:
input files: {sourceAnnotations.ToStrings}
annotations: []
last round: false
Round 3:
input files: {}
annotations: []
last round: true

This example demonstrates how tools can harvest source file annotations to produce other files. The generated files don’t have to be source files. Annotation processors may choose to generate XML descriptors, property files, shell scripts, HTML documentation, and so on.

NOTE: Some people have suggested using annotations to remove an even bigger drudgery. Wouldn't it be nice if trivial getters and setters were generated automatically? For example, the annotation

@Property private String title;

could produce the methods

public String getTitle() { return title; }
public void setTitle(String title) { this.title = title; }

However, those methods need to be added to the same class. This requires editing a source file, not just generating another file, and is beyond the capabilities of annotation processors. It would be possible to build another tool for this purpose, but such a tool would go beyond the mission of annotations. An annotation is intended as a description about a code item, not a directive for adding or changing code.
8.7 Bytecode Engineering

You have seen how annotations can be processed at runtime or at the source code level. There is a third possibility: processing at the bytecode level. Unless annotations are removed at the source level, they are present in the class files. The class file format is documented (see http://docs.oracle.com/javase/specs/jvms/se8/html). The format is rather complex, and it would be challenging to process class files without special libraries. One such library is the ASM library, available at http://asm.ow2.org.

8.7.1 Modifying Class Files

In this section, we use ASM to add logging messages to annotated methods. If a method is annotated with

```java
@LogEntry(logger=loggerName)
```

then we add the bytecodes for the following statement at the beginning of the method:

```java
Logger.getLogger(loggerName).entering(className, methodName);
```

For example, if you annotate the `hashCode` method of the `Item` class as

```java
@LogEntry(logger="global") public int hashCode()
```

then a message similar to the following is printed whenever the method is called:

```
May 17, 2016 10:57:59 AM Item hashCode
FINER: ENTRY
```

To achieve this, we do the following:

1. Load the bytecodes in the class file.
2. Locate all methods.
3. For each method, check whether it has a `LogEntry` annotation.
4. If it does, add the bytecodes for the following instructions at the beginning of the method:

```java
        ldc loggerName
        invokestatic
            java/util/logging/Logger.getLogger:(Ljava/lang/String;)Ljava/util/logging/Logger;
        ldc className
        ldc methodName
        invokevirtual java/util/logging/Logger.entering:(Ljava/lang/String;Ljava/lang/String;)V
```

Inserting these bytecodes sounds tricky, but ASM makes it fairly straightforward. We don’t describe the process of analyzing and inserting bytecodes in detail. The
important point is that the program in Listing 8.12 edits a class file and inserts a logging call at the beginning of the methods annotated with the LogEntry annotation.

For example, here is how you add the logging instructions to Item.java in Listing 8.13, where asm is the directory into which you installed the ASM library:

```
javac set/Item.java
javac -classpath .:asm/lib/\* bytecodeAnnotations/EntryLogger.java
java -classpath .:asm/lib/\* bytecodeAnnotations.EntryLogger set.Item
```

Try running
```
javap -c set.Item
```
before and after modifying the Item class file. You can see the inserted instructions at the beginning of the hashCode, equals, and compareTo methods.

```
public int hashCode();
Code:
  0: ldc #85; // String global
  2:  invokevirtual #80;
      // Method java/util/logging/Logger.getLogger:(Ljava/lang/String;)Ljava/util/logging/Logger;
  5:  ldc     #86; //String Item
  7:  ldc     #88; //String hashCode
  9:  invokevirtual #84;
      // Method java/util/logging/Logger.entering:(Ljava/lang/String;Ljava/lang/String;)V
 12:  bipush  13
 14:  aload_0
 15:  getfield        #2; // Field description:Ljava/lang/String;
 18:  invokevirtual #15; // Method java/lang/String.hashCode():I
 21:  imul
 22:  bipush  17
 24:  aload_0
 25:  getfield        #3; // Field partNumber:I
 28:  imul
 29:  iadd
 30:  ireturn
```

The SetTest program in Listing 8.14 inserts Item objects into a hash set. When you run it with the modified class file, you will see the logging messages.

```
May 17, 2016 10:57:59 AM Item hashCode
FINER: ENTRY
May 17, 2016 10:57:59 AM Item hashCode
FINER: ENTRY
May 17, 2016 10:57:59 AM Item hashCode
FINER: ENTRY
May 17, 2016 10:57:59 AM Item equals
FINER: ENTRY
[[description=Toaster, partNumber=1729], [description=Microwave, partNumber=4104]]
```
Note the call to `equals` when we insert the same item twice.

This example shows the power of bytecode engineering. Annotations are used to add directives to a program, and a bytecode editing tool picks up the directives and modifies the virtual machine instructions.

Listing 8.12  bytecodeAnnotations/EntryLogger.java

```java
class EntryLogger extends ClassVisitor
{
    private String className;

    @Override
    public MethodVisitor visitMethod(int access, String methodName, String desc, String signature, String[] exceptions)
    {
        MethodVisitor mv = cv.visitMethod(access, methodName, desc, signature, exceptions);
        return new AdviceAdapter(Opcodes.ASM5, mv, access, methodName, desc)
        {
            private String loggerName;
            public AnnotationVisitor visitAnnotation(String desc, boolean visible)
            {
                return new AnnotationVisitor(Opcodes.ASM5)
                {
                    // (Continues)
                }
            }
        }
    }

    @Override
    public MethodVisitor visitMethod(int access, String methodName, String desc, String signature, String[] exceptions)
    {
        MethodVisitor mv = cv.visitMethod(access, methodName, desc, signature, exceptions);
        return new AdviceAdapter(Opcodes.ASM5, mv, access, methodName, desc)
        {
            private String loggerName;
            public AnnotationVisitor visitAnnotation(String desc, boolean visible)
            {
                return new AnnotationVisitor(Opcodes.ASM5)
                {
                    // (Continues)
                }
            }
        }
    }
}
```

(Continues)
public void visit(String name, Object value) {
    if (desc.equals("LbytecodeAnnotations/LogEntry;") && name.equals("logger"))
        loggerName = value.toString();
}
}

public void onMethodEnter() {
    if (loggerName != null) {
        visitLdcInsn(loggerName);
        visitMethodInsn(INVOKESTATIC, "java/util/logging/Logger", "getLogger",
                        "(Ljava/lang/String;)Ljava/util/logging/Logger;", false);
        visitLdcInsn(className);
        visitLdcInsn(methodName);
        visitMethodInsn(INVOKEVIRTUAL, "java/util/logging/Logger", "entering",
                        "(Ljava/lang/String;Ljava/lang/String;)V", false);
        loggerName = null;
    }
}

/**
 * Adds entry logging code to the given class.
 * @param args the name of the class file to patch
 * @param
 */
public static void main(String[] args) throws IOException {
    if (args.length == 0) {
        System.out.println("USAGE: java bytecodeAnnotations.EntryLogger classfile");
        System.exit(1);
    }
    Path path = Paths.get(args[0]);
    ClassReader reader = new ClassReader(Files.newInputStream(path));
    ClassWriter writer = new ClassWriter(
        ClassWriter.COMPUTE_MAXS | ClassWriter.COMPUTE_FRAMES);
    EntryLogger entryLogger = new EntryLogger(writer,
        path.toString().replace("\class", "\`).replaceAll("/[\/]", "."));
    reader.accept(entryLogger, ClassReader.EXPAND_FRAMES);
    Files.write(Paths.get(args[0]), writer.toByteArray());
}
Listing 8.13 set/Item.java

```java
package set;

import java.util.*;
import bytecodeAnnotations.*;

/**
 * An item with a description and a part number.
 * @version 1.01 2012-01-26
 * @author Cay Horstmann
 */
public class Item
{
    private String description;
    private int partNumber;

    /**
     * Constructs an item.
     * @param aDescription the item's description
     * @param aPartNumber the item's part number
     */
    public Item(String aDescription, int aPartNumber)
    {
        description = aDescription;
        partNumber = aPartNumber;
    }

    /**
     * Gets the description of this item.
     * @return the description
     */
    public String getDescription()
    {
        return description;
    }

    public String toString()
    {
        return "[description=", description, ", partNumber=", partNumber, "]";
    }

    @LogEntry(logger = "com.horstmann")
    public boolean equals(Object otherObject)
    {
        if (this == otherObject) return true;
        if (otherObject == null) return false;

        // (Continues)
    }
```
```java
46 if (getClass() != otherObject.getClass()) return false;
47 Item other = (Item) otherObject;
48 return Objects.equals(description, other.description) && partNumber == other.partNumber;
49 }

50 @LogEntry(logger = "com.horstmann")
51 public int hashCode()
52 {
53 return Objects.hash(description, partNumber);
54 }

55 }

Listing 8.14 set/SetTest.java

1 package set;
2
3 import java.util.*;
4 import java.util.logging.*;
5
6 /**
7 * @version 1.02 2012-01-26
8 * @author Cay Horstmann
9 */
10 public class SetTest
11 {
12 public static void main(String[] args)
13 {
14 Logger.getLogger("com.horstmann").setLevel(Level.FINEST);
15 Handler handler = new ConsoleHandler();
16 handler.setLevel(Level.FINEST);
17 Logger.getLogger("com.horstmann").addHandler(handler);
18 Set<Item> parts = new HashSet<>();
19 parts.add(new Item("Toaster", 1279));
20 parts.add(new Item("Microwave", 4104));
21 parts.add(new Item("Toaster", 1279));
22 System.out.println(parts);
23 }
24 }

8.7.2 Modifying Bytecodes at Load Time

In the preceding section, you saw a tool that edits class files. However, it can be cumbersome to add yet another tool into the build process. An attractive
alternative is to defer the bytecode engineering until *load time*, when the class loader loads the class.

The *instrumentation API* has a hook for installing a bytecode transformer. The transformer must be installed before the *main* method of the program is called. You can meet this requirement by defining an *agent*, a library that is loaded to monitor a program in some way. The agent code can carry out initializations in a *premain* method.

Here are the steps required to build an agent:

1. **Implement a class with a method**
   
   ```java
   public static void premain(String arg, Instrumentation instr)
   ```
   
   This method is called when the agent is loaded. The agent can get a single command-line argument, which is passed in the *arg* parameter. The *instr* parameter can be used to install various hooks.

2. **Make a manifest file** `EntryLoggingAgent.mf` that sets the *Premain-Class* attribute, for example:
   
   ```manifest
   Premain-Class: bytecodeAnnotations.EntryLoggingAgent
   ```

3. **Package the agent code and the manifest into a JAR file:**
   
   ```sh
   javac -classpath .:asm/lib/* bytecodeAnnotations/EntryLoggingAgent.java
   jar cvfm EntryLoggingAgent.jar bytecodeAnnotations/EntryLoggingAgent.mf \
   bytecodeAnnotations/Entry*.class
   ```

To launch a Java program together with the agent, use the following command-line options:

```
java -javaagent:AgentJARFile=agentArgument . . .
```

For example, to run the *SetTest* program with the entry logging agent, call

```
javac set/SetTest.java
java -javaagent:EntryLoggingAgent.jar=set.Item -classpath .:asm/lib/* set.SetTest
```

The *Item* argument is the name of the class that the agent should modify.

Listing 8.15 shows the agent code. The agent installs a class file transformer. The transformer first checks whether the class name matches the agent argument. If so, it uses the *EntryLogger* class from the preceding section to modify the bytecodes. However, the modified bytecodes are not saved to a file. Instead, the transformer returns them for loading into the virtual machine (see Figure 8.3). In other words, this technique carries out “just in time” modification of the bytecodes.
Figure 8.3  Modifying classes at load time

Listing 8.15  bytecodeAnnotations/EntryLoggingAgent.java

```
package bytecodeAnnotations;

import java.lang.instrument.*;
import org.objectweb.asm.*;

/**
 * @version 1.10 2016-05-10
 * @author Cay Horstmann
 */
public class EntryLoggingAgent {
    public static void premain(final String arg, Instrumentation instr) {
        instr.addTransformer((loader, className, cl, pd, data) -> {
            if (!className.equals(arg)) return null;
            ClassReader reader = new ClassReader(data);
            ClassWriter writer = new ClassWriter(
                ClassWriter.COMPUTE_MAXS | ClassWriter.COMPUTE_FRAMES);
            EntryLogger el = new EntryLogger(writer, className);
            reader.accept(el, ClassReader.EXPAND_FRAMES);
            return writer.toByteArray();
        });
    }
}
```
In this chapter, you have learned how to

- Add annotations to Java programs
- Design your own annotation interfaces
- Implement tools that make use of the annotations

You have seen three technologies for processing code: scripting, compiling Java programs, and processing annotations. The first two were quite straightforward. On the other hand, building annotation tools is undeniably complex and not something that most developers will need to tackle. This chapter gave you the background for understanding the inner workings of the annotation tools you will encounter, and perhaps piqued your interest in developing your own tools.

In the next chapter, we’ll move on to an entirely different topic: security. Security has always been a core feature of the Java platform. As the world in which we live and compute gets more dangerous, a thorough understanding of Java security will be of increasing importance for many developers.
This page intentionally left blank
Index

Numbers
- (minus sign)
  in permissions, 521
  in policy files, 520
  in regular expressions, 129
  in URLs, 270
- (underscore)
  in native method names, 941
  in SQL, 289, 319
  in URLs, 270
, (comma)
  decimal, 372, 378, 686
  in DTDs, 165
; (semicolon)
  in classpath, 292
  in method signatures, 962
  in SQL, 293
  not needed, in annotations, 455
: (colon)
  as delimiter in text files, 63
  in classpath, 292
  in permissions, 521
  in URLs, 258
!= operator (SQL), 289
? (question mark)
  in DTDs, 165
  in e-mail URLs, 927
  in glob patterns, 112–129
  in masks, 692
  in prepared queries, 309
  in URLs, 269
/ (slash)
  ending, in codebase URLs, 516
  in method signatures, 962
  in paths, 101
  in URLs, 258
. (period)
  decimal, 372, 378, 686
  in method signatures, 962
  in regular expressions, 129–130
  in URLs, 270
leading, in file names, 516
., in paths, 102
^ (caret), in regular expressions, 129, 132
~ (tilde), in URLs, 270
' (single quote), in masks, 692
" . . ", in SQL, 289
" . . ", in XML, 146
( . . )
  in method signatures, 962
  in regular expressions, 131, 133
[ (array), type code, 87, 961
{. . .}
  in annotations, 466
  in glob patterns, 112
  in message formatting, 400–404
  in regular expressions, 132
[. . .]
  in DOCTYPE declaration, 164
  in glob patterns, 112
  in regular expressions, 129–130
  in XPath, 190
$ (dollar sign)
  in native method names, 941
  in regular expressions, 129–130, 132,
  141
${. . .}, in policy files, 521
* (asterisk)
  in DTDs, 165
  in glob patterns, 112
  in masks, 692
  in permissions, 521
  in policy files, 520
  in regular expressions, 129, 132
\ (backslash)
  in glob patterns, 112
  in paths, 56, 101
  in permissions (Windows), 520
  in regular expressions, 129–130, 141
\|, in regular expressions, 64
AbstractFormatter class
getDocumentFilter method, 689, 701
stringToValue, valueToString methods, 693, 701
AbstractListModel class, 589
AbstractProcessor class, 476
AbstractSpinnerModel class
fireStateChanged method, 706
get/setValue methods, 705, 712
getNextValue, getPreviousValue methods, 705–706, 712
AbstractTableModel class, 605
isCellEditable method, 628
accept method (ServerSocket), 242, 245
acceptChanges method (CachedRowSet), 331–332
AccessController class, getContext method, 536
Action listeners
annotating, 457–462
for text fields, 683
installing, 457
action.properties file, 449
ActionListener interface, 457
actionPerformed method, 458
@ActionListenerFor annotation, 457, 472
ActionListenerInstaller class, 457
add method
of Area, 786–788
of AttributeSet, 881
of DefaultMutableTreeNode, 642, 650
of SystemTray, 932, 936
addActionListener method
injecting into frame constructor, 458
of TrayIcon, 937
using annotations instead of, 457
addAttribute method (AttributesImpl), 232
addBatch method (Statement), 345–347
addCellEditorListener method (CellEditor), 638
addChangeListener method (JList), 737, 741
addColumn method (JTable), 617, 622
addDocumentListener method (Document), 682, 685
element method (DefaultListModel), 595
addVetoableChangeListener method (JComponent), 759
addHyperlinkListener method (EditorPane), 718
addListSelectionListener method (JFrame), 587
addRecipient method (MimeMessage), 278
addTab method (JTabbedPane), 736, 740
addTreeModelListener method (TreeModel), 672, 680
addTreeSelectionListener method (JTree), 664
addVetoableChangeListener method (JComponent), 759
AES (Advanced Encryption Standard), 568
  generating keys in, 569–574

A
\ in masks, 692
\a, \A, \ in regular expressions, 130, 132
abort method (LogimModule), 546
absolute method (ResultSet), 323, 327
AbstractCellEditor class, 629–631
isCellEditable method, 630

& (ampersand)
in CDATA sections, 149
e-mail URIs, 927
in entity references, 148
parsing, 165
&**, in regular expressions, 130
&#, &x, in character references, 148
# (number sign)
in masks, 692
in message formatting, 403–404
in URLs, 258
%(percent sign)
in fortune program, 933
in locales, 378
in SQL, 289, 319
in URLs, 270
%20, in URLs, 927
+ (plus sign)
in DTDs, 165
in regular expressions, 129, 132
in URLs, 270
< (left angle bracket)
in CDATA sections, 149
in message formatting, 404
parsing, 165
<!.. .-->, <?. .?>, <![CDATA[. . .]], in XML, 149
<.. .>, in regular expressions, 131
<= operator, 404
>, << operators (SQL), 289
== operator, in enumerations, 94
> (right angle bracket), in XML, 149
| (vertical bar)
as delimiter in text files, 63
in DTDs, 165, 167
in message formatting, 403
in regular expressions, 129, 131
\0, in regular expressions, 130
1931 CIE XYZ color specification, 836
2D graphics, 765–937
printing, 852–862
Xxx2D, Xxx2D.Float, Xxx2D.Double classes, 770–771

A
\ in masks, 692
\a, \A, \ in regular expressions, 130, 132
abort method (LoginModule), 546
absolute method (ResultSet), 323, 327
AbstractCellEditor class, 629–631
isCellEditable method, 630

AbstractDocumentFilter method, 689, 701
stringToValue, valueToString methods, 693, 701
AbstractListModel class, 589
AbstractProcessor class, 476
AbstractSpinnerModel class
fireStateChanged method, 706
get/setValue methods, 705, 712
getNextValue, getPreviousValue methods, 705–706, 712
AbstractTableModel class, 605
isCellEditable method, 628
accept method (ServerSocket), 242, 245
acceptChanges method (CachedRowSet), 331–332
AccessController class, getContext method, 536
Action listeners
annotating, 457–462
for text fields, 683
installing, 457
action.properties file, 449
ActionListener interface, 457
actionPerformed method, 458
@ActionListenerFor annotation, 457, 472
ActionListenerInstaller class, 457
add method
of Area, 786–788
of AttributeSet, 881
of DefaultMutableTreeNode, 642, 650
of SystemTray, 932, 936
addActionListener method
injecting into frame constructor, 458
of TrayIcon, 937
using annotations instead of, 457
addAttribute method (AttributesImpl), 232
addBatch method (Statement), 345–347
addCellEditorListener method (CellEditor), 638
addChangeListener method (JList), 737, 741
addColumn method (JTable), 617, 622
addDocumentListener method (Document), 682, 685
element method (DefaultListModel), 595
addVetoableChangeListener method (JComponent), 759
addHyperlinkListener method (EditorPane), 718
addListSelectionListener method (JFrame), 587
addRecipient method (MimeMessage), 278
addTab method (JTabbedPane), 736, 740
addTreeModelListener method (TreeModel), 672, 680
addTreeSelectionListener method (JTree), 664
addVetoableChangeListener method (JComponent), 759
AES (Advanced Encryption Standard), 568
generating keys in, 569–574
aes/AESTest.java, 571
aes/Util.java, 572
Affine transformations, 802
  on pixels, 843
AffineTransform class, 802–804
    constructor, 803
    getXXXInstance methods, 802–804
    setToXXX methods, 802, 804
AffineTransformOp class, 843
    constructor, 850
    TYPE_XXX fields, 843, 850
afterLast method (ResultSet), 323, 327
Agent code, 487–488
Aliases, for namespaces, 172, 197
allMatch method (Stream), 13
allocate method (ByteBuffer), 123, 125
AllPermission class, 513, 519, 523
Alpha channel, 807–811
Alpha composites, 817–818
AlphaComposite class, 809
  getInstance method, 809, 817
  &amp;, entity reference, 148
Anchor rectangle, 798
andFilter method (RowFilter), 615, 625
AnnotatedConstruct interface, 477
AnnotatedElement interface
  getAnnotation method, 457, 462, 475, 477
  getAnnotations, getDeclaredAnnotations,
    isAnnotationPresent methods, 462
  getAnnotationsByType methods, 462, 475, 477
Annotation interface
  annotationType method, 463
  equals, hashCode, toString methods, 464
  extending, 463
Annotation interfaces, 456, 462–463
  methods of, 463
  predefined, 470–475
Annotation processors, 476
  at bytecode level, 458, 481
  at runtime, 458
Annotations, 455–462
  applicability of, 472–473
  documented, 472, 474
  elements of, 456, 463, 466
  for compilation, 471
  for local variables, 467
  for managing resources, 472
  for packages, 467
  generating source code with, 477–480
  inherited, 474
marker, 465
meta, 456, 470–475
no annotations for, 468
no circular dependencies in, 466
processing tools for, 455
repeateable, 475
retaining, 472
single value, 465
source-level, 475–480
standard, 470–475
syntax of, 462–469
transient, 473
vs. Javadoc comments, 455
Antialiasing, 817–819
ANY element content (DTD), 165
anyMatch method (Stream), 13
Apache, 143, 150
Apache Derby database, 291
  connecting to, 295
  default user name/password for, 295
  driver for, 292
  populating, 305–309
  registering driver class for, 294
  starting, 293
Apache HttpClient, 274
Apache Tomcat, 349
Apollo 11, launch of, 355, 361
  &apos;, entity reference, 148
append method
  of Appendable, 54–55
  of Book, 873
  of shape classes, 776, 786
Appendable interface, 52–55
appendChild method (Node), 209, 212
Applets
  accessing browser’s cache, 261
  class loaders for, 496
  code base of, 510
  executing from secure intranet, 547
  not exiting JVM, 509
  popping up dialog boxes, 261
  running in browser, 564
  security mechanisms in, 491–579
  warning windows in, 526
appletviewer program, 515, 564
Application servers, 349, 472
Applications
  communicating with native code, 903–921
  configuring, 144–145
  data transfers between, 888
desktop, 927–932
text, 921
enterprise, 349–350
loading, 921
localizing, 408
monitoring, 487–488
paid, 499
server-side, 267–277
signing, 561–567
splash screen for, 921–926
startup plugins for, 922
updates of, 932
with internal frames, 741–744
applyPattern method (MessageFormat), 401
apply-templates element (XSLT), 224
Arc2D class, 769, 772–773
Arc2D.Double class, 785
ArcMaker class, 777
Arcs
bounding rectangle of, 770, 772
closure types of, 772–773
computing angles of, 773
Area class
add method, 786–788
subtract, intersect, exclusiveOr methods, 787–788
Areas, 786–788
ARGB (Alpha, Red, Green, Blue), 811, 837
ARRAY data type (SQL), 347–348
ArrayIndexOutOfBoundsException, 975
Arrays
from stream elements, 19
getting from a database, 347
in annotation elements, 466
in native code, 970–974
of primitive types, 973
type code for, 87, 961
type use annotations in, 467
vs. C types, 970–974
Arrays class, stream method, 8, 36
ArrayStoreException, 975
asCharBuffer method (ByteBuffer), 123
ASCII standard, 67
and native code, 950
converting native encodings to, 408
in property files, 410
in regular expressions, 133
ASM library, 481–489
ASP (Active Server Pages), 268
ATTLIST declaration (DTD), 167
attribute element (XML Schema), 174
Attribute interface, 879
category method, 881, 886
getName method, 886
implementing, 881
Attribute sets, 881
Attributes (XML)
enumerating, 154
for enumerated types, 168
in XML Schema, 174
legal, 167
names of, 146
namespace of, 198
values of, 146
accessing in XPath, 190
copying with XSLT, 225
default (DTDs), 167
normalizing, 168
vs. elements, 147–148, 168, 212
Attributes interface, getXXX methods, 204–205
AttributesSet interface, 880
add, get methods, 881, 886
remove, toArray methods, 886
AttributesImpl class, addAttribute, clear methods, 232
atZone method (LocalDate), 361
AudioPermission class, 519
auth/AuthTest.java, 534
auth/AuthTest.policy, 535
auth/jaas.config, 535
auth/SysPropAction.java, 535
Authentication, 530–546
problems of, 556–557
role-based, 537
separating from business logic, 532
through a trusted intermediary, 557–558
AuthPermission class, 520
Autoboxing, 601
AutoCloseable interface, close method, 53
Autocommit mode (databases), 344
turning off, 346
Autoflushing, 61
Autogenerated keys, 320–321
Automatic registration, of JAR files, 294
Auxiliary files, generated, 455
available method (InputStream), 49–50, 727
average method (XxxStream), 37, 39–40
AWT (Abstract Window Toolkit), 765–937
AWTPermission class, 518, 526
Books, counting words in, 2
Boolean data type (SQL), 291, 348
boolean type
  printing, 61
  type code for, 87, 961
  vs. C types, 947
  writing in binary format, 69
Bootstrap class loader, 493
Bounding rectangle, 770
boxing method (of streams), 36, 39–40
Bray, Tim, 146
Breadth-first enumerations, 659
breadFirstEnumeration method
  (DefaultMutableTreeNode), 659, 663
browse method (Desktop), 932
Browsers
  bytecode verification in, 506
  cache of, 261
  custom, 713
  forms in, 267–277
  launching from desktop applications, 927
  password-protected pages in, 261
  response page in, 268
  running applets in, 564
Buffer class, 124–126
  capacity method, 126
  clear method, 125–126
  flip method, 125–126
  hasRemaining method, 122
  limit method, 122
  mark method, 125–126
  position method, 126
  remaining method, 125–126
  reset method, 125–126
  rewind method, 125–126
BufferedImage class, 798, 834
  constructor, 835, 840
  getColorModel method, 837, 840
  getRaster method, 835, 840
  TYPE_BYTE_GRAY field, 838, 840
  TYPE_INT_ARGB field, 835–836, 840
BufferedImageOp interface, 834
  filter method, 842, 850
  implementing, 842
BufferedInputstream, BufferedOutputStream classes, 59
BufferedReader class, readline method, 63
Buffers, 124–126
  capacity of, 124
  flushing, 49, 61
  in-memory, 53
Index

1007

limits of, 124
marks in, 124
positions in, 117, 124
traversing all bytes in, 117
vs. random access, 116
BufferUnderflowException, 123
@BugReport annotation, 473
Business logic, 284
Butt cap, 788–789
Button listeners, for a dialog box, 688
ButtonFrame class, 437, 457
buttons1/ButtonFrame.java, 442
buttons2/action.properties, 452
buttons2/ButtonFrame.java, 451
buttons3/ButtonFrame.java, 460
BYTE_ARRAY class (DocFlavor), 875
Byte order mark, 407
byte type
  negative values and unsigned integers, 693
  type code for, 87, 961
  vs. C types, 947
ByteBufferOutputStream class, 98, 445
ByteBuffer class, 117, 124–126
  allocate method, 123, 125
  asCharBuffer method, 123
  get method, 117, 122
  getXXX methods, 117, 123
  order method, 118, 123
  put method, 122
  putXXX methods, 118, 123
  wrap method, 123, 125
bytecodeAnnotations/EntryLogger.java, 483
bytecodeAnnotations/EntryLoggingAgent.java, 488
Bytes, reading/writing, 48–51
C
C (char), type code, 87, 961
C programming language
  array types in, 970–974
  bootstrap class loader in, 493
calling Java methods from, 963–970
database access in, 282
embedding JVM into, 980–985
FILE* type in, 51
header file generating, 942
invalid pointers in, 940
strings in, 949
types, vs. Java types, 947
\c, in regular expressions, 130
C++ programming language
  accessing JNI functions in, 950
  array types in, 971
  embedding JVM into, 980–985
  exceptions in, 975
  for native methods, 940, 943
  invalid pointers in, 940
  pointers to dummy classes in, 965
Cache, 261
Cached row sets, 329–334
CachedRowSet interface, 329–333
  acceptChanges method, 331–332
  execute method, 330, 332
  get/setPageSize method, 330, 332
  get/setTableName method, 331–332
  nextPage method, 330, 332
  populate method, 330, 332
  previousPage method, 332
CachedRowSetImpl class, 329
Caesar cipher, 498–499
Calendar class, 351
  formatting objects of, 386
call escape (SQL), 319
call method (CompilationTask), 445, 448
Call stack, during permission checking, 513
callback interface, 445
callback interface, 538
callbackHandler interface, handle method, 545
callNonvirtualXXXMethod functions (C), 966, 969
callStaticXXXMethod functions (C), 964–965, 970
callXXXMethod functions (C), 963, 969
cancelCellEditing method (CellEditor), 631–632, 638
cancelRowUpdates method (ResultSet), 325, 328
canImport method (TransferHandler), 912, 918
canInsertImage method (ImageWriter), 827, 834
capacity method (Buffer), 126
Carriage return character, displaying, 154
Cascading windows, 744–745
Casts, type use annotations in, 468
Catalogs, 342

CDATA declaration (DTD), 167–168
CDATA sections (XML), 149
Cell editors (Swing), 628–629
  custom, 629–638
Cell renderers (Swing)
  for lists, 595–599
  for tables, 609, 626
  for trees, 661–664
CellEditor interface
  add/removeCellEditorListener methods, 638
  cancelCellEditing method, 631–632, 638
  getCellEditorValue method, 629, 631–632, 638
  isCellEditable method, 638
  shouldSelectCell method, 631, 638
  stopCellEditing method, 631–632, 638
Cells (Swing)
  editing, 628–629
  rendering, 626–638
  selecting, 612–614, 626
Certificates, 531, 553–556
  and Java Plug-in, 563
  managing, 560–561
  publishing fingerprints of, 555
  root, 563
  set of, 510
  signing, 558–561
CertificateSIGNER class, 559
CGI (Common Gateway Interface), 268
Change listener interface, 737
changeUpdate method (DocumentListener), 682, 685
Channels, 250
  for files, 117
Channels class
  newInputStream method, 256
  newOutputStream method, 250, 257
char type
  type code for, 87, 961
  vs. C types, 947
CHAR_ARRAY class (DocFlavor), 875
Character classes, 129
CHARACTER data type (SQL), 291, 348
Character encoding, 60, 67–69
  converting to ASCII, 408
  platform, 68–69, 404
Character references (XML), 148
CharacterData interface, getData method, 153, 162
Characters
  antialiasing, 817
  differences between, 394
  escaping, 64, 129
  in regular expressions, 133
  normalizing, 394
  outlines of, 805
  printing, 61
  turning to upper/lower case, 690
  writing in binary format, 69
  characters method (ContentHandler), 200, 204
charAt method (String), 9
CharBuffer class, 53–54, 124
  get, put methods, 124
CharSequence interface, 54, 134
  charAt, length methods, 55
  chars method, 36
  codePoints method, 36, 40
  splitting by regular expressions, 6
  subSequence, toString methods, 55
Charset class
  defaultCharset method, 68, 406
  forName method, 69
Checkboxes (Swing), 626
  checked attribute (HTML, XML), 146
  Checked exceptions, 533
checkError method (PrintWriter), 61–62
checkExit method (SecurityManager), 509, 512
checkLogin method (SimpleLoginModule), 538
checkPermission method (SecurityManager), 512–513,
  523–524
checkRead method (SecurityManager), 513
Child elements (XML), 147
  getting node list of, 154
  namespace of, 197
Child nodes (Swing), 639
  adding, 642
  connecting lines for, 645–646
children method (TreeNode), 658
choice element (XML Schema), 173
choice keyword (message formatting), 403
Church, Alonzo, 355
Cipher class, 567–569
  doFinal method, 568, 571, 573, 575
  getInstance method, 567, 573
  getXX_SIZE methods, 573
  init method, 573
  update method, 568, 571, 573, 575
  XXX_MODE modes, 568
CipherInputStream class, read method, 575
CipherOutputStream class, 574
  constructor, 575
  flush, write methods, 575
Ciphers
generating keys, 569–574
public keys in, 575–579
streams for, 574–575
symmetric, 567–569
Circular dependencies, in annotations, 466
Class
class
getClassLoader method, 493, 503
getFields method, 674
getProtectionDomain method, 514
implementing AnnotatedElement, 457
Class files, 492
corrupted, 504–508
encrypted, 498–499
format of, 481
loading, 492
locating, 444
modifying, 481–486
portability of, 407
producing on disk, 445
storing in databases, 444
transformers for, 487
verifying, 504–508
Class loaders, 450, 492–508
as namespaces, 496
bootstrap, 493
text, 495–496
creating, 509
extension, 493
hierarchy of, 494–496
multiple, serving, 245–248
Class parameter (MIME types), 892
Class path hell, 493
Class references, in native code, 958
.class file extension, 492
Classes
adding validation to, 92
annotating, 456, 466, 470
descriptions of, 86
externalizable, 87
inheritance trees of, 660
loading, 492–494
locales supported by, 376
nonserializable, 92
protection domains of, 512
resolving, 492
separate for each web page, 496
serializable, 80–81
versioning, 95–98
Classifier functions, 28
ClassLoader class, 493
defineClass, findClass methods, 498, 503
extending, 497
getParent, getSystemClassLoader methods, 503
loadClass method, 496, 498
ClassLoader inversion, 494
classLoader/Caesar.java, 502
classLoader/ClassLoaderTest.java, 499
CLASSPATH environment variable, 493
CLEAR composition rule, 810
clear method
of AttributesImpl, 232
of Buffer, 125–126
clearParameters method (PreparedStatement), 316
Client/server applications, 285
Clients
configuring securely, 566
connecting to servers, 236–238
multiple, serving, 245–248
clip method (Graphics2D), 767, 805–807, 854
Clipboard, 887–903
accessing, 509
capabilities of, 888
transferring:
images, 894
object references, 902
objects, 898–902
text, 888–892
Clipboard class, 888
addFlavorListener method, 894
constructor, 903
get/setContents methods, 889, 891
getAvailableDataFlavors method, 894
getData method, 892
isDataFlavorAvailable method, 891
ClipboardOwner interface
implementing, 888
lostOwnership method, 888, 892
Clipping, 766, 805–807
Clipping region
printing, 854
setting, 767
CLOB data type (SQL), 291, 348
Clob interface, 316
get/setCharacterStream methods, 316, 318
getSubString method, 316–317
length method, 317
CLOBs (character large objects), 316
creating empty, 318
placing in database, 316
clone method, 81, 98
Cloning, 98–100
close method
of AutoCloseable, 53
of Connection, 299, 301, 350
of FileLock, 128
of Flushable, 52
of InputStream, 49–50
of OutputStream, 51
of ProgressMonitor, 723, 731
of ResultSet, 301
of ServerSocket, 245
of SplashScreen, 926
of Statement, 300–301
of XMLStreamWriter, 222
Closeable interface, 52
close method, 52, 54
flush method, 52
Closed nonleaf icons, 648, 661
closeEntry methods (ZipXxxStream), 77–78
closeOnCompletion method (Statement), 300
closePath method (Path2D), 776, 786
Closure types, 772
cmd shell, 406
Code base, 510, 516
Code generation, annotations for, 455–462, 471
Code source, 510
Code units (UTF-16), in regular expressions, 130
Codebreakers, The (Kahn), 498
codePoints method (CharSequence), 36, 40
CodeSource class, getXxx methods, 514
Collation, 393–400
collation/CollationTest.java, 396
CollationKey class, compareTo method, 400
Collator class, 393
compare, equals methods, 399
get/setDecomposition methods, 399
get/setStrength methods, 399
getAvailableLocales method, 399
getCollationKey method, 395, 399
getInstance method, 399
collect method (Stream), 19, 22, 35
collecting/CollectingIntoMaps.java, 25
collecting/CollectingResults.java, 20
collecting/DownstreamCollectors.java, 31
Collection interface
parallelStream method, 2–5, 41–46
stream method, 2–5
Collections
processing, 2–5
vs. streams, 3
Collections class, sort method, 394
Collector interface, 19
Collectors class
counting method, 29
groupingBy method, 28–31
groupingByConcurrent method, 43
joining method, 20, 23
mapping method, 30, 33
maxBy, minBy methods, 30, 33
partitioningBy method, 28, 31
summarizingXxx methods, 20, 23
summingXxx methods, 29, 33
toCollection method, 20, 23
toConcurrentMap method, 25, 27
toList method, 23
toMap method, 24–27
toSet method, 20, 23, 29
Collectors, downstream, 29–33
Color chooser, 629, 898–902
Color class, 797
constructor, 842
getRGB method, 842
translating values into pixel data, 838
Color space conversions, 844
ColorConvertOp class, 843–844
ColorModel class, 838
dataElements method, 842
getRGB method, 837, 842
Colors
components of, 807
composing, 808–811
dithering, 818
interpolating, 797–798
negating, 844
solid, 766
Columns (databases)
accessing by number, in result set, 299
names of, 285
number of, 334
Columns (Swing)
accessing, 610
adding, 617
detached, 601
hiding, 617–625
names of, 606
rendering, 609–610
resizing, 602, 611–612
selecting, 612
com.sun.rowset package, 329
com.sun.security.auth.module package, 532
Combo boxes
   editing, 629
traversal order in, 704
Comments (XML), 149
commit method
   of Connection, 344–346
   of LoginModule, 546
Commit or revert behavior, 687–688
commitEdit method (JFormattedTextField), 688, 701
Comparable interface, 615
Comparators, 614
compare method (Collator), 399
to method
   of CollationKey, 400
   of Comparable, 615
   of String, 393
CompiledScript interface, 437
Compiler
   annotations for, 471
   invoking, 443
   just-in-time, 980
   producing class files on disk, 445
Compiler API, 443–454
compiler/ByteArrayJavaClass.java, 446
compiler/CompilerTest.java, 452
compiler/MapClassLoader.java, 454
compiler/StringBuilderJavaSource.java, 446
Complex types, 172
complexType element (XML Schema), 173
Composite, 809
composite/CompositeComponent.java, 814
composite/CompositeTestFrame.java, 812
composite/Rule.java, 815
Composition rules, 766–767, 807–817
Computer Graphics: Principles and Practice
   (Foley), 775, 809, 836
concat method (Stream), 11
Confidential information, transferring, 567
Configuration files, 126
connect method
   of Socket, 239
   of URLConnection, 259, 262, 266
Connection interface
close method, 299, 301, 350
commit method, 344–346
createBlob, createClob methods, 316, 318
createStatement method, 298–299, 321, 326, 344
get/setAutoCommit methods, 346
getMetaData method, 333, 342
getWarnings method, 304
prepareStatement method, 310, 315, 322, 326
releaseSavepoint method, 345–346
rollback method, 344–346
setSavepoint method, 346
Connections (databases)
closing, 302
   using row sets after, 329
debugging, 279
pooling, 350
starting new threads, 246
console method (System), 406
Constructive area geometry operations, 786
Constructor class, 457
Constructors
   annotating, 466
   invoking from native code, 965–966
type use annotations in, 468
containsAll method (Collection), 525
Content types, 260
ContentHandler class, 199, 201
characters method, 200, 204
start/endDocument methods, 204
start/endElement methods, 200–204
Context class loader, 495–496
Control points
   dragging, 777
   of curves, 774
   of shapes, 776
convertXxxIndexToModel methods (JTable), 614, 623
Convolution operation, 844
ConvolveOp class, 843–845
   constructor, 851
Coordinate system
   custom, 767
   translating, 856
Coordinate transformations, 799–804
Cubic curves, 774–775
CubicCurve2D class, 769, 774
CubicCurve2D.Double class, 785
Currencies, 384–385
  available, 385
  formatting, 378–383, 691
  identifiers for, 384
Currency class, 384–385
  getAvailableCurrencies method, 385
  getCurrencyCode method, 385
  getDefaultFractionDigits method, 385
getInstance method, 384–385
getSymbol method, 385
toString method, 385
Cursor, drop shapes of, 904
curveTo method (Path2D.Float), 775, 786
Custom editors, 629
Custom formatters, 693–703
Cut and paste, 887–903
Cygwin, 944
  compiling invocation API, 984
  OpenSSL in, 560
D
\d, \D, in regular expressions, 131
Dashed lines, dash pattern, dash phase, 790
Data flavors, 892–894
Data sources, 349
Data transfer, 887–903
  between Java and native code, 888
  classes and interfaces for, 888
  in Swing, 904–921
Data types
  codes for, 87, 961
  in Java vs. C, 947
  mangling names of, 961
  print services for, 874–875
database.properties file, 305, 349
DatabaseMetaData interface, 333–343
  getJDBCVersion methods, 343
  getMaxConnection method, 343
  getMaxStatements method, 301, 343
  getSQLStateType method, 302
  getTables method, 333, 342
  supportsBatchUpdates method, 345, 347
  supportsResultSet methods, 323, 328
Databases, 281–350
  accessing, in C language, 282
  autoincrementing keys in, 320
  avoiding duplication of data in, 287
  batch updates for, 345–347
  caching prepared statements, 311
  changing data with SQL, 290
  connections to, 292, 294–297, 306
  closing, 301, 306
  in web and enterprise applications, 349–350
  pooling, 350
drivers for, 283–284
error handling in, 346
integrity of, 344
LOBs in, 316–318
metadata for, 333–343
modifying, 330
native storage for XML in, 349
numbering columns in, 299
outer joins in, 319
populating, 305–309
saving objects to, 474
scalar functions in, 319
schemas for, 342
setting up parameters in, 330
starting, 293
stored procedures in, 319
storing:
   class files, 444
   logins, 537
   structure of, 285, 333
   synchronization of, 331
tools for, 334
truncated data from, 303
URLs of, 292
DataFlavor class, 888, 892–894
   constructor, 893
   getHumanPresentableName method, 894
   getType method, 893
   getRepresentationClass method, 894
   imageFlavor constant, 895
   isMimeTypeEqual method, 894
   javaFileListFlavor constant, 914
DataInput interface, 70
   readFully method, 71
   readUTF method, 70–71
   readXXX methods, 70–71, 81
   skipBytes method, 71
DataInputStream class, 51, 56
DataOutput interface, 69
   writeUTF method, 70, 72
   writeXXX methods, 69, 72, 81
DataOutputStream class, 51
DataSource interface, 349
DataTruncation class, 303
getLength, getParameter methods, 304
getTransferSize method, 305
Date and Time API, 351–370
   and legacy code, 369
   Date class, 87, 369
      formatting objects of, 386
      read/writeObject methods, 93
   DATE data type (SQL), 291, 318, 348
   Date pickers, 705
   DateEditor class, 712
dataFilter method (RowFilter), 615, 625
DateFormat class, 387
   DateFormat class, 365–68, 372, 385–393
   lenient, 691
   literals for, 318
   local, 355–358
   parsing, 367
   DateTimeFormatter class, 365–68, 385–393
      and legacy classes, 369
      format method, 365, 392
      getLocalizedXXX methods, 365, 385, 392
      ofPattern method, 367
      parse method, 367
      toFormat method, 367
      withLocale method, 365, 392
   DateTimeSyntax class, 882
   Daylight savings, 361–365
   DayOfWeek enumeration, 357
   getDisplayName method, 365, 386
   dayOfWeekInMonth method (TemporalAdjusters), 359
   DECIMAL data type (SQL), 291, 348
   Decimal separators, 372, 378, 686
   Declaration annotations, 466–468
   decode method (URLDecoder), 277
   Decryption key, 498
   default statement, 462
Debugging
   in JNI, 980
   JDBC-related problems, 296
   mail connections, 279
   of locales, 377
   policy files, 564
   streams, 11
   with telnet, 233–236
Declaring objects, 988
Declaring properties, 466
Declaring variables, 466
**Index**

<table>
<thead>
<tr>
<th>Class/Method/Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultCellEditor</td>
<td>class, 654</td>
</tr>
<tr>
<td></td>
<td>constructor, 638</td>
</tr>
<tr>
<td></td>
<td>variations, 629</td>
</tr>
<tr>
<td>defaultCharset</td>
<td>method (Charset), 68, 406</td>
</tr>
<tr>
<td>DefaultFormatter</td>
<td>class extending, 693</td>
</tr>
<tr>
<td></td>
<td>get/setOverwriteMode methods, 692, 702</td>
</tr>
<tr>
<td></td>
<td>toString method, 691</td>
</tr>
<tr>
<td>DefaultHandler</td>
<td>class, 201</td>
</tr>
<tr>
<td>DefaultListModel</td>
<td>class, 594</td>
</tr>
<tr>
<td></td>
<td>add/removeElement methods, 595</td>
</tr>
<tr>
<td>DefaultMutableTreeNode</td>
<td>class, 641, 659–661</td>
</tr>
<tr>
<td></td>
<td>add method, 642, 650</td>
</tr>
<tr>
<td></td>
<td>constructor, 650</td>
</tr>
<tr>
<td></td>
<td>depthFirstEnumeration method, 659</td>
</tr>
<tr>
<td></td>
<td>pathFromAncestorEnumeration method, 660</td>
</tr>
<tr>
<td></td>
<td>setAllowsChildren, setAsksAllowsChildren methods, 648, 650</td>
</tr>
<tr>
<td></td>
<td>setAsksAllowsChildren method, 648</td>
</tr>
<tr>
<td></td>
<td>XxxxFirstEnumeration, XxxxOrderEnumeration methods, 659, 663</td>
</tr>
<tr>
<td>defaultPage</td>
<td>method (PrinterJob), 861</td>
</tr>
<tr>
<td>DefaultRowSorter</td>
<td>class, 626–627</td>
</tr>
<tr>
<td></td>
<td>isCellEditable method, 628</td>
</tr>
<tr>
<td>DefaultTableModel</td>
<td>class, 650, 673</td>
</tr>
<tr>
<td></td>
<td>automatic notification by, 652</td>
</tr>
<tr>
<td></td>
<td>getPathToRoot method, 653</td>
</tr>
<tr>
<td></td>
<td>insertNodeInto method, 652, 658</td>
</tr>
<tr>
<td></td>
<td>isLeaf method, 648</td>
</tr>
<tr>
<td></td>
<td>modeChanged method, 652, 658</td>
</tr>
<tr>
<td></td>
<td>nodesChanged method, 658</td>
</tr>
<tr>
<td></td>
<td>reload method, 653, 658</td>
</tr>
<tr>
<td></td>
<td>removeNodeFromParent method, 652, 658</td>
</tr>
<tr>
<td></td>
<td>defaultWriteObject method (ObjectOutputStream), 92</td>
</tr>
<tr>
<td></td>
<td>defineClass method (ClassLoader), 498, 503</td>
</tr>
<tr>
<td>Delayed formatting</td>
<td>888</td>
</tr>
<tr>
<td>delete method</td>
<td>107–108</td>
</tr>
<tr>
<td>@Deprecated annotation</td>
<td>470–471</td>
</tr>
<tr>
<td>Depth-first enumerations</td>
<td>659</td>
</tr>
<tr>
<td></td>
<td>depthFirstEnumeration method (DefaultMutableTreeNode), 659, 663</td>
</tr>
<tr>
<td>derbyclient.jar file</td>
<td>292</td>
</tr>
<tr>
<td>DES (Data Encryption Standard)</td>
<td>568</td>
</tr>
<tr>
<td>Desktop class</td>
<td>921, 927–932</td>
</tr>
<tr>
<td></td>
<td>browse method, 932</td>
</tr>
<tr>
<td></td>
<td>edit method, 927, 932</td>
</tr>
<tr>
<td></td>
<td>getDesktop method, 927, 931</td>
</tr>
<tr>
<td></td>
<td>isDesktopSupported method, 927, 931</td>
</tr>
<tr>
<td></td>
<td>isSupported method, 927, 931</td>
</tr>
<tr>
<td></td>
<td>mail method, 932</td>
</tr>
<tr>
<td></td>
<td>open method, 927, 932</td>
</tr>
<tr>
<td></td>
<td>print method, 927, 932</td>
</tr>
<tr>
<td>Desktop applications, launching</td>
<td>927–932</td>
</tr>
<tr>
<td>Desktop panes</td>
<td>741–760</td>
</tr>
<tr>
<td></td>
<td>cascading/tiling, 744–748</td>
</tr>
<tr>
<td>desktopApp/DesktopAppFrame.java</td>
<td>928</td>
</tr>
<tr>
<td>DesktopManager class</td>
<td>751</td>
</tr>
<tr>
<td>DestroyJavaVM function (C)</td>
<td>981, 985</td>
</tr>
<tr>
<td>Device coordinates</td>
<td>799</td>
</tr>
<tr>
<td>Diagnostic interface</td>
<td>444</td>
</tr>
<tr>
<td></td>
<td>getKind method, 448</td>
</tr>
<tr>
<td></td>
<td>getMessage method, 449</td>
</tr>
<tr>
<td></td>
<td>getSource method, 448</td>
</tr>
<tr>
<td></td>
<td>getXxxNumber methods, 449</td>
</tr>
<tr>
<td>DiagnosticCollector class</td>
<td>444</td>
</tr>
<tr>
<td></td>
<td>constructor, 448</td>
</tr>
<tr>
<td></td>
<td>getDiagnostics method, 448</td>
</tr>
<tr>
<td>DiagnosticListener interface</td>
<td>444</td>
</tr>
<tr>
<td>DialogCallbackHandler class</td>
<td>538</td>
</tr>
<tr>
<td>Dialogs</td>
<td>744–748</td>
</tr>
<tr>
<td></td>
<td>in internal frames, 750</td>
</tr>
<tr>
<td></td>
<td>validating input fields before closing, 688</td>
</tr>
<tr>
<td></td>
<td>digest method (MessageDigest), 549–550</td>
</tr>
<tr>
<td>Digital signatures</td>
<td>546–567</td>
</tr>
<tr>
<td></td>
<td>verifying, 553–556</td>
</tr>
<tr>
<td>Direct buffers</td>
<td>973</td>
</tr>
<tr>
<td>Directories</td>
<td>564</td>
</tr>
<tr>
<td></td>
<td>creating, 105–106</td>
</tr>
<tr>
<td></td>
<td>current, 114</td>
</tr>
<tr>
<td></td>
<td>hierarchical structure of, 639</td>
</tr>
<tr>
<td></td>
<td>printing all subdirectories of, 113</td>
</tr>
<tr>
<td></td>
<td>traversing, 110–114</td>
</tr>
<tr>
<td></td>
<td>user’s working, 56</td>
</tr>
<tr>
<td>DirectoryStream interface</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>displayMessage method (TrayIcon), 937</td>
</tr>
</tbody>
</table>
distinct method (Stream), 11–12, 43
Dithering, 818
dividedBy method (Instant, Duration), 353
dnd/SampleComponents.java, 907
dnd/SwingDnDTest.java, 907
dndImage/imageListDnDFrame.java, 915
doAs, doAsPrivileged methods (Subject), 532–533, 536
Doc interface, 874
DocAttribute interface, 879
  implementing, 881
  printing attributes of, 882–885
DocAttributeSet interface, 880–881
DocFlavor class, 874–875, 878
DocPrintJob interface
  getAttributes method, 886
  print method, 877
DOCTYPE declaration (DTD), 164
  including in output, 210
Document interface
  addDocumentListener method, 682, 685
  createXxx methods, 209, 212
  getDocumentElement method, 150, 161
  getLength method, 685
  getText method, 685
  implementing, 681
Document filters, 688–690
Document flavors, for print services, 874–875
Document listeners, 682
DocumentBuilder class
  newDocument method, 209, 212, 227
  parse method, 160
  setEntityResolver method, 164, 170
  setErrorHandler method, 170
DocumentBuilderFactory class
  is/setIgnoringElementContentWhitespace methods, 169, 171
  is/setNamespaceAware methods, 174, 198–199, 201, 209
  is/setValidating method, 171
  newDocumentBuilder method, 150, 160, 209
  newInstance method, 150, 160
  setNamespaceAware method, 209
  setValidating method, 169
@Documented annotation, 470, 472, 474
DocumentEvent interface, getDocument method, 685
DocumentFilter class, 689
  insertString method, 688–689, 702
  remove method, 702
  replace method, 689, 702
DocumentListener interface, XxxUpdate methods, 682, 685
doFinal method (Cipher), 568, 571, 573, 575
DOM (Document Object Model) parser, 149–150, 199
  supporting PUBLIC identifiers in, 164
DOM (Document Object Model) tree accessing with XPath, 190–196
  analyzing, 152–154
  building, 208–222
  writing, 210–211
don/TreeViewer.java, 155
DOMResult class, 227, 232
DOMSource class, 213, 226
DOUBLE data type (SQL), 291, 348
double type
  printing, 61
    type code for, 87, 961
    vs. C types, 947
  writing in binary format, 69
DoubleBuffer class, 124
Double-clicking, on list components, 585
doubles method (Random), 37, 40
DoubleStream interface, 36–41
  average, max, min, sum methods, 37, 40
  boxed method, 36, 40
  mapToDouble method, 36
  of method, 39
  summaryStatistics method, 37, 40
  toArray method, 36, 40
DoubleSummaryStatistics class, 20, 23, 37, 41
doubleValue method (Number), 378
Downstream collectors, 29–33
Drag and drop, 903–921
  file lists in, 914
  in Swing components, 904–921
  moving vs. copying with, 903
  visual feedback for, 912
Drag sources, 903
  configuring, 909–912
draw method (Graphics2D), 767–769, 787
Drawing
  shapes, 766–769
  simple, 765
drawXxx methods (Graphics), 769
DriverManager class, 294
  getConnection method, 295, 297, 306, 350
  setLogWriter method, 296
Drop actions, 903–904
Drop locations, 914
DROP TABLE statement (SQL), 295
  executing, 298, 300
  in batch updates, 345
Drop targets, 903, 912–921
DropLocation class (of JList)
  getIndex method, 914, 920
  isInsert method, 915, 920
DropLocation class (of JTable), getXxx, isInsertXxx
  methods, 920
DropLocation class (of JTable), getDropPoint method, 920
DropLocation class (of JTable), isInsertXxx methods, 920
DropLocation class (of JTable), getPath method, 914, 920
DropLocation class (of JEditorPane), isInsertXxx methods, 920
DropLocation class (of JEditorPane, getDropPoint method, 920
DropLocation class (of JTree), getChildIndex method, 920
DropLocation class (of JTree), getPath method, 914, 920
DropLocation class (of TransferHandler), getDropPoint method, 920
DSAs (Digital Signature Algorithm), 551–552
DST, DST_XX composition rules, 810
DTDHandler class, 201
DTDs (Document Type Definitions), 163–171
  element content in, 165–166
  entities in, 168
  external, 164
  in XML documents, 147, 164
  unambiguous, 166
  URLs for, 164
Duration class
  arithmetic operations, 353
  between method, 352
  immutable, 354
  txXxx methods, 353
Dynamic links, 981
Dynamic web pages, 449–454

E
  \e, \E, in regular expressions, 130
Echo servers, 244
Eclipse, startup plugins for, 922
Edge detection, 845
edit method (Desktop), 927, 932
Editor pane (Swing), 681, 712–719
  edit mode of, 714
  loading pages in separate threads, 715
Editor pane/EditorPaneFrame.java, 716
Editors, custom, 629
element element (XML Schema), 172
Element interface, 477
  getAttribute method, 154, 161, 176
  getSimpleName method, 477
getTagName method, 151, 161, 198
setAttribute, setAttributeNS methods, 209, 212
ELEMENT element content (DTD), 165–166
Elements (XML)
  child, 147
    accessing in XPath, 191
    namespace of, 197
  constructing, 209
  counting, in XPath, 191
  empty, 146
  getting node list of, 154
  legal attributes of, 167
  names of, 151, 198
  root, 147, 172
  trimming whitespace in, 153
  vs. attributes, 147–148, 168, 212
Ellipse2D class, 769
Ellipses, bounding rectangle of, 770
E-mails, 927
  launching from desktop applications, 927
  sending, 277–280
  terminating lines in, 278
employee/Employee.c, 959
employee/Employee.java, 959
employee/EmployeeTest.java, 958
empty method
  of Optional, 16
  of Stream, 5, 8
EMPTY element content (DTD), 165
Empty tags (XML), 146
encode method (URLEncoder), 277
Encryption, 567–579
  final block padding in, 568
  of class files, 498–499
end method (Matcher), 135, 137, 140–141
End cap styles, 788–790
End points, 774
End tags (XML), 146
endDocument method (ContentHandler), 204
endElement method (ContentHandler), 200–204
End-of-line character. See Line feed
Enterprise applications, 349–350
Enterprise JavaBeans (EJBs), 285
  hot deployment of, 496
Entity references (XML), 148, 168
Entity resolvers, 150
ENTITY, ENTITIES attribute types (DTDs), 167–168
EntityResolver class, 201
  resolveEntity method, 164, 170
entries method (ZipFile), 80
Index
Index

File pointers, 72
File systems, POSIX-compliant, 109
file: URI scheme, 257, 517
file.encoding system property, 68
file.separator property, 521
FileChannel class
lock, tryLock methods, 126–128
open, map methods, 117, 122
FileHandler class, 406
FileInputStream class, 55–59, 513, 523, 726 constructor, 58
getChannel method, 121
read method, 48
fileKey method (BasicFileAttributes), 109
FileLock class
close method, 128
isShared method, 127
FileNotFoundException, 273
FileOutputStream class, 55–59 constructor, 59
getChannel method, 121
FilePermission class, 511, 517
FileReader class, 513
Files
accessing, 509
channels for, 117
closing, 110
configuration, 126
copying, 106
counting lines in, 727
creating, 105–106
deleting, 107
drag and drop of, 904–905
encrypting/decrypting, 574
filtering, 112
by file suffixes, 825
generated automatically, 455, 475
handling, 445
hierarchical structure of, 639
I/O modes of, 76
in desktop applications, 927
locating, 444
locking, 126–128
memory-mapped, 116–128
moving, 106
random-access, 72–76
vs. buffered, 116
reading, 104–105
as a string, 104
by one byte, 48–51
numbers from, 56
permissions for, 523
total number of bytes in, 73
traversing, 111–115
with multiple images, 825–834
writing, 104–105
Files class, 100, 104–114
copy method, 106, 108
createXxx methods, 105–106
delete method, 107–108
deleteIfExists method, 108
exists method, 108–109
find method, 111
getBytes method, 104
getOwner method, 108
isXxx methods, 108–109
lines method, 6, 8
list method, 110
move method, 106, 108
newDirectoryStream method, 111, 114
newXxxStream, newXxxFiles methods, 104–105
readAllXxx methods, 104
readAttributes method, 109
size method, 108–109
walk method, 110
write method, 104
Files class, walkFileTree method, 112–114
FileSystem class, getPath method, 115–116
FileSystems class, newFileSystem method, 115–116
FileNode class, and legacy classes, 369
FileVisitor interface, 113–114
visitFile, visitFileFailed methods, 112
xxxVisitDirectory, visitXXX methods, 112
fill method (Graphics2D), 767–768, 787
Filling, 766–767, 797
fillXxx methods (Graphics), 769
filter method
of BufferedImageOp, 842, 850
of Stream, 3–4, 9–10, 13
FilteredRowSet interface, 329
Filters
for images, 842–851
for numbers, 615
for streams, 726
for table rows, 615–617
for user input, 688–690
glob patterns for, 112
implementing, 616
FilterXXXStream classes, 57
Final block padding, 568
find method
  of Files, 111
  of Matcher, 137, 140
findAny method (Stream), 13
findClass method (ClassLoader), 498, 503
findClass function (C), 957, 960, 965
findColumn method (ResultSet), 301
findXxx methods (Stream), 13
Fingerprints, 86, 546–567
different for a class and its objects, 89
fireStateChanged method (AbstractSpinnerModel), 706
first method (ResultSet), 323, 327
firstDayOfYearXxx methods (TemporalAdjusters), 359
#FIXED attribute (DTD), 167
Fixed-size patterns, 692
Fixed-size records, 73–74
flatMap method of Optional, 16–19
  of Stream, 10
FlavorListener interface, flavorsChanged method, 893–894
flavormap.properties file, 887
flip method (Buffer), 125–126
FLOAT data type (SQL), 291, 348
float type
  printing, 61
type code for, 87, 961
  vs. C types, 947
  writing in binary format, 69
floatBuffer class, 124
Floating-point numbers, 372, 378–383, 691
flush method
  of CipherOutputStream, 575
  of Closeable, 52
  of Flushable, 52, 55
  of OutputStream, 49, 51
Flushable interface, 52–53
close method, 52
  flush method, 52, 55
fn keyword (SQL), 319
Focus listeners, 683
Folder icons, 648, 661
Font render context, 805
Fonts
  antialiasing, 817, 819
displaying, 596
forEach method (Stream), 19, 22
Forest (Swing), 639, 647
forLanguageTag method (Locale), 377
format method
  of DateTimeFormatter, 365, 392
  of Format, 402
  of MessageFormat, 401–402
  of NumberFormat, 379, 383
Formatters
  and locales, 373
custom, 693–703
  installing, 689
  supported by JFormattedTextField, 691–693
Formatting
  copy and paste, 887
delayed, 888
  of currencies, 691
  of dates, 372, 385–393
  of messages, 400–404
  of numbers, 372, 378–383, 691
formatting/formatting.java, 368
Forms, processing, 267–277
forName method (Charset), 69
fortune program (UNIX), 933
ForwardingJavaFileManager class
  constructor, 449
getFileForOutput method, 449
getJavaFileForOutput method, 445, 450
fprintf function (C), 963
Fractals, 838
Frame class, 450
Frames (Swing)
cascading/tiling, 744–748
closing, 748
dialogs in, 750
dragging, 751–760
grabbers of, 742
icons for, 743
internal, 741–752
managing on desktop, 751
positioning, 744
resizable, 745
selected, 744, 747
states of, 745
visibility of, 744
  with two nested split panes, 732
from method (Instant), 369
FROM statement (SQL), 288
FTP (File Transfer Protocol), 262
ftp: URI scheme, 257, 262, 713
Function interface, identity method, 24
@FunctionalInterface annotation, 470
getAvailableCurrencies
getAutoCreateRowSorter
getAuthority
getAttribute
getAscent
getAllowUserInteraction
getAutoCommit
getArrayLength
getAnnotation
getAllByName
getAttribute
getAs
getAnnotations
getAnnotationsByType
generateKey
@Generated
generate
GeneralPath
garbage collection
G
getBounds method (SplashScreen), 926
getBundle method (ResourceBundle), 409–412
getByClassName method (InetAddress), 240–241
getByteArrayElements method (Blob), 316–317
getByteArrayElements function (C), 972–974
getByteArrayRegion function (C), 973–974
getBooleanField function (C), 960
getBounds method (SplashScreen), 926
getBundle method (ResourceBundle), 409–412
getByClassName method (InetAddress), 240–241
getByteArrayElements function (C), 972–974, 989
getByteArrayRegion function (C), 973–974
getBytes method
of Blob, 316–317
der of files, 104
getCandidateLocales method (ResourceBundle.Control), 410
getCategory method (Attribute), 881, 886
getCellEditorValue method (CellEditor), 629, 631–632, 638
getCertificates method (CodeSource), 514
getchannel method (FileInputStream, RandomAccessFile), 121
getchar method (ByteBuffer), 117, 123
getCharacterStream method (Blob), 316, 318
getCharArrayElements function (C), 972–974
getCharArrayRegion function (C), 973–974
genCharContent method (SimpleJavaFileObject), 445, 449
genCharField function (C), 960
getchild method (TreeModel), 154, 672–674, 680
getchildAt method (TreeNode), 658
getchildCount method
of TreeModel, 672–674, 679
der of TreeNode, 658
genchildIndex method (JTree.DropLocation), 920
getchildNodes method (Node), 151, 161
getClassLoader method (Class), 493, 503
genClipboard method (Graphics), 806, 854
genCloset method (ResultSet), 316–317
genCodeSource method (ProtectionDomain), 514
genCollationKey method (Collator), 395, 399
getAvailableDataFlavors method (Clipboard), 894
getAvailableLocales method
of Collator, 399
of NumberFormat, 376, 379, 383
getAverage method (XxxSummaryStatistics), 20, 23, 41
getBackground method (JList), 596, 599
genBinaryStream method (Blob), 316–317
genBlob method (ResultSet), 316–317
getBlockSize method (Cipher), 573
getBooleanArrayElements function (C), 972–974
getBooleanArrayRegion function (C), 973–974
getBooleanField function (C), 960
genBounds method (SplashScreen), 926
genBundle method (ResourceBundle), 409–412
getByClassName method (InetAddress), 240–241
getByteArrayElements function (C), 972–974
getByteArrayRegion function (C), 973–974
getBytes method
of Blob, 316–317
der of files, 104
genCandidateLocales method (ResourceBundle.Control), 410
genCategory method (Attribute), 881, 886
genCellEditorValue method (CellEditor), 629, 631–632, 638
genCertificates method (CodeSource), 514
genChannel method (FileInputStream, RandomAccessFile), 121
genchar method (ByteBuffer), 117, 123
genCharacterStream method (Blob), 316, 318
getCharArrayElements function (C), 972–974
getCharArrayRegion function (C), 973–974
genCharContent method (SimpleJavaFileObject), 445, 449
genCharField function (C), 960
getchild method (TreeModel), 154, 672–674, 680
getchildAt method (TreeNode), 658
getchildCount method
of TreeModel, 672–674, 679
der of TreeNode, 658
genchildIndex method (JTree.DropLocation), 920
getchildNodes method (Node), 151, 161
genClassLoader method (Class), 493, 503
genClipboard method (Graphics), 806, 854
genCloset method (ResultSet), 316–317
genCodeSource method (ProtectionDomain), 514
genCollationKey method (Collator), 395, 399
getAvailableDataFlavors method (Clipboard), 894
getAvailableLocales method
of Collator, 399
of NumberFormat, 376, 379, 383
getAverage method (XxxSummaryStatistics), 20, 23, 41
getBackground method (JList), 596, 599
genBinaryStream method (Blob), 316–317
genBlob method (ResultSet), 316–317
getBlockSize method (Cipher), 573
getBooleanArrayElements function (C), 972–974
getBooleanArrayRegion function (C), 973–974
getBooleanField function (C), 960
genBounds method (SplashScreen), 926
genBundle method (ResourceBundle), 409–412
getByClassName method (InetAddress), 240–241
getByteArrayElements function (C), 972–974
getByteArrayRegion function (C), 973–974
getBytes method
of Blob, 316–317
der of files, 104
genCandidateLocales method (ResourceBundle.Control), 410
genCategory method (Attribute), 881, 886
genCellEditorValue method (CellEditor), 629, 631–632, 638
genCertificates method (CodeSource), 514
genChannel method (FileInputStream, RandomAccessFile), 121
genchar method (ByteBuffer), 117, 123
genCharacterStream method (Blob), 316, 318
getCharArrayElements function (C), 972–974
getCharArrayRegion function (C), 973–974
genCharContent method (SimpleJavaFileObject), 445, 449
genCharField function (C), 960
getchild method (TreeModel), 154, 672–674, 680
getchildAt method (TreeNode), 658
getchildCount method
of TreeModel, 672–674, 679
der of TreeNode, 658
genchildIndex method (JTree.DropLocation), 920
getchildNodes method (Node), 151, 161
genClassLoader method (Class), 493, 503
genClipboard method (Graphics), 806, 854
genCloset method (ResultSet), 316–317
genCodeSource method (ProtectionDomain), 514
genCollationKey method (Collator), 395, 399
getDecomposition method (Collator), 399
default method (locale), 376–377
defaultEditor method (JTable), 637
defaultFractionDigits method (Currency), 385
defaultName method (NameCallback), 545
defaultRenderer method (JTable), 627, 637
descent method (TextLayout), 807
desktop method (Desktop), 927, 931
desktopPane method (JInternalFrame), 759
diagnostics method (DiagnosticCollector), 448
GetDirectBufferXxx functions (C), 973
gDisplayCountry, gDisplayLanguage methods
(locale), 377
gDisplayName method
of DayOfWeek, 365, 386
of Locale, 376–377, 379
of Month, 365, 386
document method (DocumentEvent), 685
documentElement method (Document), 150, 161
documentFilter method (AbstractFormatter), 689, 701
gDouble method
of ByteBuffer, 117, 123
of ResultSet, 299, 301
GetDoubleArrayElements function (C), 972–974
GetDoubleArrayRegion function (C), 973–974
GetDoubleField function (C), 956, 960
gDoXxx methods (URLConnection), 259, 265
gDropAction method (TransferSupport), 919
gDropLocation method (TransferSupport), 914, 919
gDropPoint method (TransferHandler.DropLocation), 920
gElementAt method (ListModel), 589, 593
gEnclosedElements method (TypeElement), 477
gEngineXxx methods (ScriptEngineManager), 431
gEntry method (Zipfile), 80
gErrorCode method (SQLException), 302, 304
gErrorStream method (HttpURLConnection), 273, 277
gErrorWriter method (ScriptContext), 434
gExpiration method (URLConnection), 260, 263, 267
gExtensions method (ScriptEngineFactory), 431
GetFieldID function (C), 957, 960
getFields method (Class), 674
getFileForOutput method (ForwardingJavaFileManager), 449
gFileName method (StackTraceElement), 103
gFilePointer method (RandomAccessFile), 72, 76
gFileSuffixes method (ImageReaderWriterSpi), 833
gFillsViewportHeight method (JTable), 604
getFirstChild method (Node), 153, 161
getFloat method (ByteBuffer), 117, 123
GetFloatArrayElements function (C), 972–974
GetFloatArrayRegion function (C), 973–974
GetFloatField function (C), 960
getFocusLostBehavior method (FormattedTextField), 688, 701
getFontRenderContext method (Graphics2D), 805, 807
getForeground method (JList), 596, 599
getFormatNames method (ImageReaderWriterSpi), 833
getFragment method (URL), 258
getFrameIcon method (JInternalFrame), 759
getHeaderXxx methods (URLConnection), 260–262, 267
getHeight method
  of ImageReader, 826, 833
  of PageFormat, 854, 861
getHost method (URL), 258
getHostXxx methods (InetAddress), 241
gethour method (Date and Time API), 360, 363
getHumanPresentableName method (DataFlavor), 894
getIconAt method (JTabbedPane), 740
getIdentifier method (Entry), 625
getIfModifiedSince method (URLConnection), 266
getImage method (TrayIcon), 936
getImageableXxx methods (PageFormat), 855, 861–862
getImageURL method (SplashScreen), 926
getImageXxxByXxx methods (Image10), 824, 832
getIndex method
  of DataTruncation, 304
  of JList.DropLocation, 914, 920
  of JTextComponent.DropLocation, 921
getIndexOfChild method (TreePath), 672, 680
getInputStream method
  of Socket, 237–238, 242
  of URLConnection, 260, 267, 271, 273
  of ZipFile, 80
getInstance method
  of AlphaComposite, 809, 817
  of Cipher, 567, 573
  of Collator, 399
  of Currency, 384–385
  of KeyGenerator, 574
  of Locale, 393
  of MessageDigest, 548–550
getInt method
  of ByteBuffer, 117, 123
  of ResultSet, 299, 301
GetIntArrayElements function (C), 972–974
GetIntArrayRegion function (C), 973–974
getIntegerInstance method (NumberFormat), 689
getInterface method (Invokerable), 436
GetIntField function (C), 956, 960, 989
getInvalidCharacters method (MaskFormatter), 703
generateFileForOutput method
  (ForwardingJavaFileManager), 445, 450
getJavaFileObjectsFromXxx methods
  (StandardJavaFileManager), 448
get jdbcXxxVersion methods (DatabaseMetaData), 343
getkeys method (ResourceBundle), 412–413
getkind method (Diagnostic), 448
ggettext method (Locale), 377
getLastChild method (Node), 154, 161
getLastModified method (URLConnection), 260, 263, 267
getLastPathComponent method (TreePath), 651, 658
getLastSelectedPathComponent method (JTree), 652, 657
getLayoutOrientation method (JList), 587
getleading method (TextLayout), 807
getlength method
  of Attributes, 204
  of Document, 685
  of NamedNodeMap, 162
  of NodeList, 151, 162
getlineNumber method
  of Diagnostic, 449
  of SAXParseException, 171
getlistCellRendererComponent method
  (ListCellRenderer), 595–599
getLocale method (MessageFormat), 402
getlocalHost method (InetAddress), 240–241
getlocalName method
  of Attributes, 204
  of Node, 198–199
  of XMLStreamReader, 208
getlocation method (CodeSource), 514
getlong method (ByteBuffer), 117, 123
getlongArrayElements function (C), 972–974
getlongArrayRegion function (C), 973–974
getlongField function (C), 960
getMax method (XxxSummaryStatistics), 20, 23, 41
getMaxConnections method (DatabaseMetaData), 343
getMaximum method (JProgressBar), 730
getMaxStatements method (DatabaseMetaData), 301, 343
getMessage method (Diagnostic), 449
getMetaData method
  of Connection, 333, 342
  of ResultSet, 334, 343
getMethodCallSyntax method (ScriptEngineFactory), 435
getMemento method (DefaultFormatter), 198–199
getMDM of (DefaultFormatter), 653
getMetadata method (DataTruncation), 304
getMimeType method (CachedRowSet), 332
getMinimum method (DataTruncation), 304
getParent method
  of ClassLoader, 503
  of Path, 103
  of TreeNode, 658, 660
getParentNode method (Node), 161
getPassword method
  of PasswordCallback, 546
  of RowSet, 331
getPath method
  of FileSystem, 115–116
  of JTree.DropLocation, 914, 920
  of TreeSelectionEvent, 671
  of URL, 258
getPaths method (TreeSelectionEvent), 665, 671
getPathToRoot method (DefaultTreeModel), 653
getPercentInstance method (NumberFormat), 378, 383, 691
getPixel, getPixels methods (Raster), 836, 841
getPlaceholder, getPlaceholderCharacter methods (MaskFormatter), 703
getPointCount method (ShapeMaker), 776
getPopupMenu method (TrayIcon), 936
getPort method (URL), 258
getPreferredSize method (JComponent), 596–597, 685
getPreviousSibling method (Node), 161
getPreviousValue method (AbstractSpinnerModel), 705–706, 712
get Principals method (Subject), 536
getPrintable method (Book), 873
getPrinterJob method (PrinterJob), 852, 861
getPrintService method (StreamPrintServiceFactory), 878–879
getNumberInstance
getNextException
getNamespaceURI
getObject
getNextWarning
getNextValue
getNextSibling
getNextEntry
getNewValue
getNano
getNames
getName
getMoreResults
getMethodCallSyntax
getMetaData
getModel
getMnemonicAt
getMinute
getMinimum
getMin
getMIMETypes
getMimeTypes
getMimeType
getMinutes method (Date and Time API), 360, 363
getMnemonic method (JTabbedPane), 741
of Entry, 625
of JList, 595
getMonth, getMonthValue methods (Date and Time API), 356, 363
getMoreResults method (Statement), 320
getName method
  of Attribute, 886
  of Field, 674
  of NameCallback, 545
  of Permission, 525, 530
  of Principal, 537
  of PrintService, 874
  of UnixPrincipal, 531
  of XMLStreamReader, 208
  of ZipEntry, 79
  of ZipFile, 80
getNames method (ScriptEngineFactory), 431
getNamespaceURI method (Node), 198–199
getName method (Date and Time API), 360, 363
getAddress method (PropertyChangeEvent), 750, 759
getNextEntry method (ZipInputStream), 77–78
getNextException method (SQLException), 302, 304
getNextSibling method (Node), 154, 161
g getNextValue method (AbstractSpinnerModel), 705–706, 712
g getNextWarning method (SQLException), 304
g getNodeXXX methods (Node), 154, 161, 198
g getNumberOfInstance method (NumberFormat), 378, 383, 691
g getNumXXX methods (ImageReader), 826, 833
g getobject method
  of ResourceBundle, 411–412
  of ResultSet, 299, 301
GetObjectArrayElement function (C), 972–973
GetObjectClass function (C), 957–958
GetObjectField function (C), 956, 960
getOffset method (ZonedDateTime), 363
gen orientation method (PageFormat), 862
gen OriginatingProvider method
  of ImageReader, 824, 833
  of ImageWriter, 834
genOutputSize method (Cipher), 573
genOutputStream method
  of Socket, 238, 242
  of URLConnection, 260, 267, 270
gen OverwriteMode method (DefaultFormatter), 702
genOwner method (Files), 108
genPageCount method (Banner), 863
genPageSize method (CachedRowSet), 332
genParameter method (DataTruncation), 304
genParent method
  of ClassLoader, 503
  of Path, 103
  of TreeNode, 658, 660
genParentNode method (Node), 161
getPassword method
  of PasswordCallback, 546
  of RowSet, 331
getPath method
  of FileSystem, 115–116
  of JTree.DropLocation, 914, 920
  of TreeSelectionEvent, 671
  of URL, 258
getPaths method (TreeSelectionEvent), 665, 671
getPathToRoot method (DefaultTreeModel), 653
getPercentInstance method (NumberFormat), 378, 383, 691
getPixel, getPixels methods (Raster), 836, 841
getPlaceholder, getPlaceholderCharacter methods (MaskFormatter), 703
getPointCount method (ShapeMaker), 776
getPopupMenu method (TrayIcon), 936
getPort method (URL), 258
genPreferredSize method (JComponent), 596–597, 685
genPreviousSibling method (Node), 161
genPreviousValue method (AbstractSpinnerModel), 705–706, 712
gen Principals method (Subject), 536
genPrintable method (Book), 873
genPrinterJob method (PrinterJob), 852, 861
genPrintService method (StreamPrintServiceFactory), 878–879
getPrompt method
  of NameCallback, 545
  of PasswordCallback, 546
getPropertyName method (PropertyChangeEvent), 759
getProtectionDomain method (Class), 514
getPrototypeCellValue method ([List], 593
getQName method (Attribute), 204
getQualifiedName method (TypeElement), 477
getQuery method (URI), 258
gerReader method (BufferedImage), 835, 840
gerReaderXXX methods (ImageID), 825, 832
gerReadTimeout method (URLConnection), 266
gerRepresentationClass method (DataFlavor), 894
gerRequestProperties method (URLConnection), 266
gResultSet method (Statement), 300
gRGB method
  of Color, 842
  of ColorModel, 837, 842
gRoot method
  of Path, 103
  of TreeModel, 154, 672–674, 679
gRotateInstance method (AffineTransform), 802–803
gRow method
  of JTable.DropLocation, 920
  of ResultSet, 323, 327
gRowCount method (TableModel), 605–606, 608
gRowXXX methods (JTable), 622
gSavepointXXX methods (savepoint), 347
gScaleInstance method (AffineTransform), 802–803
gSecond method (Date and Time API), 360, 363
getSelectedComponent method (JTabbedPane), 740
getSelectedIndex method (JTabbedPane), 737, 740
getSelectedValue method ([List], 584, 587
getSelectedValuesList method ([List], 585, 587
getSelectionMode method ([List], 587
getSelectionModel method (JTable), 622
gSelectionPath method (JTree), 651, 657, 665, 671
gSelectionPaths method (JTree), 665, 671
gSelectionXXX methods ([List], 596, 599
gShearInstance method (AffineTransform), 802, 804
gShort method (ByteBuffer), 117, 123
GetShortArrayElements function (C), 972–974
GetShortArrayRegion function (C), 973–974
GetShortField function (C), 960
gSimpleInstance method ([Element], 477
getSize method
  of listModel, 589, 593
  of ZipEntry, 79
getSource method (Diagnostic), 448
getSourceActions method (TransferHandler), 909, 912
getSourceDropActions method (TransferHandler), 919
getSplashScreen method (SplashScreen), 923, 926
gSQLState method (SQLException), 302, 304
gStateType method (SQLException), 302
gStandardFileManager method (JavaCompiler), 447
GetStaticFieldID, GetStaticXXXField functions (C), 960–961
GetStaticMethodID function (C), 964, 970
getStrength method (Collator), 399
getString method
  of JProgressBar, 730
  of ResourceBundle, 410, 412
  of ResultSet, 299, 301
getStringArray method (ResourceBundle), 412
getStringChars, GetStringLength functions (C), 953
getStringRegion, GetStringUTFRegion, GetStringUTFLength
  functions (C), 952
GetStringUTFChars function (C), 951–952, 954, 989
getStringValue method (Entry), 625
getSubmethod (LoginContext), 536
getSubString method (Clob), 316–317
getSum method (XxxSummaryStatistics), 23, 41
GetSuperclass function (C), 1001
getSymbol method (Currency), 385
getSystemClassLoader method (ClassLoader), 503
getSystemClipboard method (Toolkit), 888, 891
getSystemJavaCompiler method (ToolProvider), 443
getSystemTray method (SystemTray), 932, 936
getTabComponentAt, getTabLayoutPolicy methods
  (JTabbedPane), 741
getTabCount method (JTabbedPane), 740
getTableCellEditorComponent method (TableCellEditor),
  629, 631, 638
getTableCellRendererComponent method
  (TableCellRenderer), 626, 637
getTableName method (CachedRowSet), 332
getTables method (DatabaseMetaData), 333, 342
getTagname method ([Element], 151, 161, 198
getTask method (JavaCompiler), 444, 448
Getter/setter pairs. See Properties
gsetText method
  of Document, 685
  of XMLEventReader, 208
gtextInstance method (DateFormat), 691
gTree method
  of JTree, 740
getTag method (JTabbedPane), 740
getTooltip method (TrayIcon), 936
getTransferable method (TransferSupport), 913
getTransferData method (Transferable), 892
getTransferDataFlavors method (Transferable), 894
getTransferSize method (DataTruncation), 305
getTranslateInstance method (AffineTransform), 802, 804
getTrayIconSize method (SystemTray), 936
getTreeCellRendererComponent method (TreeCellRenderer), 662–663
getType method of Field, 674 of ResultSet, 323, 326
getUpdateCount method (Statement), 300, 320
getURI method (Attribute), 204
getAddress method of HyperlinkEvent, 715, 719 of RowSet, 331
getUseCaches method (URLConnection), 266
getUserDropAction method (TransferSupport), 919
getUserInfo method (URI), 258
getValidCharacters method (MaskFormatter), 703
getValue method of AbstractSpinnerModel, 705, 712 of Attributes, 205 of Copies, 882 of Entry, 625 of JFormattedTextField, 687, 701 of JProgressBar, 730 of JSpinner, 704, 710 of WIN32RegKey, 987–988 getValueAt method (TableModel), 605–606, 608 getValueContainsLiteralCharacters method (MaskFormatter), 703 getValueCount method (Entry), 625 getValueIsAdjusting method (ListSelectionEvent), 584
getVendorName, getVersion methods (IIOServiceProvider), 825, 833
getVisibleRowCount method (List), 587
getWarnings method (Connection, ResultSet, Statement), 304
getWidth method of ImageReader, 826, 833 of PageFormat, 854, 861
getWriter method (ScriptContext), 434
getWriterXXX methods (ImageIO), 825, 832
getYear method (Date and Time API), 356, 363

GIF format, 823 animated, 825, 921 image manipulations on, 846 printing, 874 GlassFish server, 349 GMail, 278–279 Gnome (GNU Object Model Environment) drag and drop in, 904 supporting API operations in, 927 Gnu C compiler, 943–944 Gödel’s theorem, 504 GradientPaint class, 797 constructor, 797–798 cyclic parameter, 798
group keyword, 516, 532, 564
Graphic Java™ (Geary), 600, 639 Graphics class, 765
drawXxx, fillXxx methods, 769
get/setClip methods, 805–806, 854
Graphics2D class, 765–937
clip method, 767, 805–807, 854
draw method, 767–769, 787
fill method, 767–768, 787
getFontRenderContext method, 805, 807
rotate, scale, shear methods, 800, 804
setComposite method, 767, 809, 816
setPaint method, 767, 797–798
setRenderingHint, setRenderingHints methods, 766, 817–819, 823
setStroke method, 767, 788, 796
setTransform method, 802, 804
translate method, 800, 804, 864
GregorianCalendar class, toZonedDateTime method, 369
Grid bag layout, 175–179
GridBagConstraints class, 175
GridBagLayout class, 175–189
GridBagPane class, 179
Groovy programming language, 430–431, 438–439

group method (Watcher), 135, 140–141
groupCount method (Watcher), 141
groupingBy method (Collectors), 28–31
groupingByConcurrent method (Collectors), 43
GSS-API, 579
&gt;, entity reference, 148
GUI (Graphical User Interface), scripting events for, 437–442
H

\( h \), \( \mathcal{H} \), in regular expressions, 131
Half-closing connections, 249
Handbook of Applied Cryptography, The (Menezes et al.), 551
handle method (CallbackHandler), 545
handleGetObject method (ResourceBundle), 412–413
Handles (Swing), 644, 661
hash/Digest.java, 549
hashCode method of Annotation, 464
of Permission, 523
HashXXXAttributeSet classes, 852, 880
Haskell programming language, 430
hasMoreElements method (Enumeration), 987–989
hasNext method (XMLStreamReader), 208
hasRemaining method (Buffer), 122
Header information, from server, 259
Headers (Swing tables)
  rendering, 627
  scrolling, 601
helloNative/HelloNative.c, 943
helloNative/HelloNative.h, 942
helloNative/HelloNative.java, 941
helloNative/HelloNativeTest.java, 945
Hex editors
  creating class files in, 505
  modifying bytecodes with, 507
Hidden commands, in XML comments, 149
HORIZONTAL_SPLIT value (JSplitPane), 732
HORIZONTAL_WRAP value (JList), 583
Hosts
  and IP addresses, 239–241
  local, 240
Hot deployment, 496
HTML (HyperText Markup Language)
  attributes in, 148
  displaying with JEditorPane, 681, 712–719
  end and empty tags in, 146
  forms in, 268
  generating from XML files, 222–225
  mixing with JSP, 449
  printing, 874
  vs. XML, 145–146
HTMLDocument class, 681
HTTP (Hypertext Transfer Protocol), 285
  request headers in, 261–262
  using SSL over, 579
http: URI scheme, 257, 517, 713
https: URI scheme, 257, 579
HttpURLConnection class, getErrorStream method, 273, 277
HyperlinkEvent class, getURL method, 715, 719
HyperlinkListener interface, hyperlinkUpdate method, 715, 718
Hyperlinks, 713
I

I (int), type code, 87, 961
I/O streams. See Input streams, Output streams
IANA (Internet Assigned Numbers Authority), 361
IBM, 143, 150
IBM DB2 database, 291
IBM437 encoding, 406
ICC profiles, 836
Icons
  for frames, 743
  in column headers, 627
  in table cells, 626
  in trees, 648, 661
  one-touch expand, 732
  tray, 932
ID, IDREF, IDREFS attribute types (DTDs), 167–168
identity method (Function), 24
Identity (do-nothing) transformation, 213
Identity values, 34
IDs, uniqueness of, 168, 178
IETF BCP 47, 377
ifPresent method (Optional), 14–15, 41
IIImage class, 827, 834
IIServiceProvider class, getVendorName, getVersion methods, 825, 833
IllegalAccessException, 674
IllegalArgumentException, 207, 706, 710, 712, 976
IllegalStateException, 24, 826, 914
Imageable area, 855
imageFlavor method (DataFlavor), 895
InputStream class, “seek forward only” of, 825
ImageIO class
  createImageXXXStream methods, 825, 832
  determining image type, 823–824
  getImageXXXByXXX methods, 824, 832
  getReaderXXX, getWriterXXX methods, 825, 832
  read, write methods, 823, 831
imageIO/ImageIOFrame.java, 828
ImageOutputStream interface, 827
Incremental rendering, 834
Indexed color model, 844
indexOfComponent, indexOfTab methods (JTabbedPane), 737, 740
indexOfTabComponent method (JTabbedPane), 741
IndexOutOfBoundsException, 826
InetAddress class, 239–241
getXxx methods, 240–241
InetAddress/InetAddressTest.java, 240
InetSocketAddress class, isUnresolved method, 256
Infinite trees, 675
Inheritance trees, 303
@Inherited annotation, 471, 474
init method
  of Cipher, 573
  of KeyGenerator, 574
Initialization blocks, for shared libraries, 947
initialize method
  of LoginModule, 546
  of SimpleLoginModule, 538
INPUT_STREAM class (DocFlavor), 875
Input streams, 48–59
  and Unicode, 48
  as input source, 150
  buffered, 57–59
  byte processing in, 57
  byte-oriented, 48
  chaining, 57
  closing, 49
  encoding for, 60
  filters for, 55–59
  hierarchy of, 51–55
  keeping open, 249
  monitoring progress of, 726–731
  objects in, 80–100
  total number of bytes in, 727
Input validation mask, 685
InputSource class, 171
InputStream class, 48–51, 52
  available method, 49–50, 727
  close method, 49–50
  mark, markSupported methods, 50
  markSupported method, 50
  read method, 48–50
  reset method, 50
  skip method, 50
InputStreamReader class, 60, 726
InputVerifier class, verify method, 690
INSERT statement (SQL), 290
  and autogenerated keys, 321
INSERT statement (SQL) (continued)
executing, 298, 300, 316
in batch updates, 345
vs. methods of ResultSet, 325
insertModelInto method (DefaultTreeModel), 652, 658
insertRow method (ResultSet), 325, 328
insertString method (DocumentFilter), 688–689, 702
insertTab method (JTabbedPane), 736, 740
insertUpdate method (DocumentFilter), 682, 685
InsertNodeInto method (DefaultTreeModel), 652, 658
insertRow method (ResultSet), 325, 328
insertString method (DocumentFilter), 688–689, 702
Instant class, 352
and legacy classes, 369
arithmetic operations, 353
from method, 369
immutable, 354
now method, 352
Instrumentation API, 487
int type
printing, 61
storing, 70
type code for, 87, 961
vs. C types, 947
writing in binary format, 69
IntBuffer class, 124
INTEGER data type (SQL), 291, 348
Integers
ranges of, 36
supported locales for, 686
validating input of, 686
IntegerSyntax class, 882
@interface declaration, 456, 462
Interfaces
accessing script classes with, 436
annotating, 466, 470
implementing in script engines, 435
Internal frames (Swing), 741–752
cascading/tiling, 744–748
closing, 748
dialogs in, 750
dragging, 751–760
grabbers of, 742
icons for, 743
managing on desktop, 751
positioning, 744
resizable, 745
selected, 744
states of, 745
visibility of, 744
internalFrame/DesktopFrame.java, 752
InternalFrameListener interface, internalFrameClosing method, 751
Internationalization, 371–428
Interpolation, 843
for gradients, 797–798
strategies of, 843
when transforming images, 818, 820, 843
Interruptible sockets, 250–257
interruptible/InterruptibleSocketTest.java, 251
intersect method (Area), 787–788
ints method (Random), 37, 40
IntStream interface, 36–41
average, max, min, sum methods, 37, 39
boxed method, 36, 39
mapToInt method, 36
of, range, rangeClosed methods, 36, 38
summaryStatistics method, 37, 39
toArray method, 36, 38
IntSummaryStatistics class, 20, 23, 37, 41
tintValue method (Number), 379
Invalid pointers (C, C++), 940
InvalidPathException, 101
Invocable interface, 435
getInterface method, 436
invokeXXX methods, 435–436
Invocation API, 980–985
invocation/InvocationTest.c, 981
IOException, 237, 716
IP addresses, 235, 239–241
validating, 693–694
IPP (Internet Printing Protocol) 1.1, 885
IPv6 addresses, 239
isAfter, isBefore methods (Date and Time API), 356, 360, 363
isAfterLast method (ResultSet), 324, 327
isAnnotationPresent method (AnnotatedElement), 462
isAssignableFrom function (C), 989, 1001
isBeforeFirst method (ResultSet), 324, 327
isCanceled method (ProgressMonitor), 723, 731
isCellEditable method
of AbstractCellEditor, 630
of AbstractTableModel, 628
ISO 3166–1 country codes, 374
ISO 4217 currency identifiers, 384
ISO 639–1 language codes, 374
ISO 8601 format, 318, 471
ISO 8859–1 standard, 60, 68
isOneTouchExpandable method (JSplitPane), 734
isOutputShutdown method (Socket), 249
isParseIntegerOnly method (NumberFormat), 383
isPresent method (Optional), 15–16
iSQL-Viewer program, 334
isReadable method (Files), 108–109
isRegularFile method
  of BasicFileAttributes, 109
  of Files, 108–109
isResizable method (JInternalFrame), 758
isSelected method (JInternalFrame), 747, 758
isShared method (FileLock), 127
isStartElement method (XMLStreamReader), 208
isStringPainted method (JProgressBar), 730
isSupported method
  of Desktop, 927, 931
  of SystemTray, 932, 936
isSymbolicLink method
  of BasicFileAttributes, 109
  of Files, 108–109
isUnresolved method (InetSocketAddress), 256
isValidating method
  of DocumentBuilderFactory, 171
  of SAXParserFactory, 203
isVisible method (JInternalFrame), 759
isWhiteSpace method (XMLStreamReader), 208
isWritable method (XMLStreamReader), 208
isZero method (Instant, Duration), 353
item method
  of NodeList, 151, 162, 169
Iterable interface, 111
iterate method (Stream), 5, 8, 12, 36
Iterator interface, 298
iterator method
  of BaseStream, 19–20
  of SQLException, 302, 304
Iterators, 19

J
J (long), type code, 87, 961
JAAS (Java Authentication and Authorization Service), 530–546
  configuration files in, 531, 536
  login modules in, 537–546
jaas/jaas.config, 545
jaas/JAASTest.java, 544
jaas/JAASTest.policy, 545
jaas/SimpleCallbackHandler.java, 543
jaas/SimpleLoginModule.java, 541
jaas/SimplePrincipal.java, 540
jaas.config file, 533
JAR files
  automatic registration in, 294
  class loaders in, 495
  code base of, 510
  for plugins, 494
  manifest of, 77, 921
  resources in, 408
  signing, 555–556
jar: URI scheme, 257
jarray type (C), 988
jarsigner program, 555–556, 563
JarInputStream classes, 77
Java programming language
  deployment directory of, 564, 566
  internationalization support in, 371
  platform-independent, 70
  security of, 510–514
  vs. SQL, 311
Java 2D API, 765–937
  features supported in, 766
  rendering pipeline, 766–768
Java EE (Java Platform, Enterprise Edition), 285
Java Plug-in, loading signed code, 563
Java program
  -javaagent option, 487
  jdbc.drivers property in, 294
  -noverify option, 506
  security managers in, 516
  -splash option, 921
Java Rich Internet Applications Guide, 566
Java™ Virtual Machine Specification, The
  (Lindholm/Yellin), 506
java.awt.AlphaComposite API, 817
java.awt.BasicStroke API, 796
java.awt.Color API, 842
java.awt.datatransfer API, 888
java.awt.datatransfer.Clipboard API, 891–892, 894, 903
java.awt.datatransfer.ClipboardOwner API, 892
java.awt.datatransfer.DataFlavor API, 893–894
java.awt.datatransfer.FlavorListener API, 894
java.awt.datatransfer.Transferable API, 892, 894
java.awt.Desktop API, 931–932
java.awt.dnd API, 904
java.awt.font.TextLayout API, 807
java.awt.geom API, 92
java.awt.geom.AffineTransform API, 803–804
java.awt.geom.Area API, 788
java.awt.geom.CubicCurve2D.Double API, 785
java.awt.geom.Area API, 788
java.awt.geom.CubicCurve2D.Double API, 785
java.awt.geom.GeneralPath API, 785
java.awt.geom.Path2D API, 786
java.awt.geom.Path2D.Float API, 786
java.awt.geom.QuadCurve2D.Double API, 785
java.awt.geom.RoundRectangle2D.Double API, 785
java.awt.GradientPaint API, 798
java.awt.Graphics API, 806
java.awt.Graphics2D API, 768, 796, 798, 804, 807, 816, 823
java.awt.image.AffineTransformOp API, 850
java.awt.image.BufferedImage API, 840
java.awt.image.BufferedImageOp API, 850
java.awt.image.ByteLookupTable API, 850
java.awt.image.ColorModel API, 842
java.awt.image.ConvolveOp API, 851
java.awt.image.Kernel API, 851
java.awt.image.LookupOp API, 850
java.awt.image.RescaleOp API, 850
java.awt.image.ShortLookupTable API, 851
java.awt.image.WritableRaster API, 841
java.awt.print.Book API, 873
java.awt.print.PageFormat API, 861–862
java.awt.print.Printable API, 860
java.awt.print.PrinterJob API, 861, 873
java.awt.RenderingHints API, 823
java.awt.SplashScreen API, 926
java.awt.SystemTray API, 936
java.awt.TexturePaint API, 799
java.awt.Toolkit API, 891
java.awt.TrayIcon API, 936–937
java.beans.PropertyChangeEvent API, 759
java.beans.PropertyVetoException API, 760
java.beans.VetoableChangeListener API, 759
java.io.BufferedReader API, 59
java.io.BufferedInputStream API, 59
java.io.Closeable API, 54
java.io.DataInputStream API, 71
java.io.DataOutputStream API, 72
java.io.File API, 103
java.io.File.separator constant, 56
java.io.FileInputStream API, 58, 121
java.io.FileOutputStream API, 59, 121
java.io.Flushable API, 55
java.io.InputStream API, 49–50
java.io.ObjectInputStream API, 85
java.io.ObjectOutputStream API, 85
java.io.OutputStream API, 50–51
java.io.PrintWriter API, 62
java.io.PushbackInputStream API, 59
java.io.RandomAccessFile API, 76, 121
java.lang, java.lang.annotation packages, annotations in, 470
java.lang.annotation.Annotation API, 463–464
java.lang.Appendable API, 55
java.lang.CharSequence API, 40, 55
java.lang.Class API, 503, 514
java.lang.ClassLoader API, 503
java.lang.Readable API, 55
java.lang.reflect.AnnotatedElement API, 462
java.lang.SecurityManager API, 513
java.lang.System API, 946
java.lang.Thread API, 503
java.net package
  socket connections in, 238
  supporting IPv6 addresses in, 240
  URLs vs. URIs in, 257
java.net.HttpURLConnection API, 277
java.net.InetAddress API, 241
java.net.InetSocketAddress API, 256
java.net.ServerSocket API, 245
java.net.Socket API, 238–239, 249
java.net.URL API, 265
java.net.URLClassLoader API, 503
java.net.URLConnection API, 265–267
java.net.URLDecoder API, 277
java.net.URLEncoder API, 277
java.nio package, 247, 250
  direct buffers in, 973
  memory mapping in, 117
java.nio.Buffer API, 122, 126
java.nio.ByteBuffer API, 122–123
java.nio.channels.Channels API, 256–257
java.nio.channels.FileChannel API, 122, 128
java.nio.channels.FileLock API, 128
java.nio.channels.SocketChannel API, 256
java.nio.CharBuffer API, 124
java.nio.file.attribute.BasicFileAttributes API, 109
java.nio.file.Files API, 8, 104–106, 108–109, 114
java.nio.file.FileSystem API, 116
java.nio.file.Filesystems API, 116
java.nio.file.Path API, 103
java.nio.file.Paths API, 103
java.nio.file.SimpleFileVisitor API, 115
java.policy file, 515, 565
java.security package, 492, 547
java.security file, 515
java.security.CodeSource API, 514
java.security.MessageDigest API, 550
java.security.Permission API, 530
java.security.Principal API, 537
java.security.PrivilegedAction API, 536
java.security.PrivilegedExceptionAction API, 537
java.security.PrivilegedDomain API, 514
java.sql.Blob API, 317
java.sql.Clob API, 317–318
java.sql.Connection API, 299, 304, 315, 318, 326, 342, 346
java.sql.DatabaseMetaData API, 328, 342–343, 347
java.sql.DriverManager API, 297
java.sql.PreparedStatement API, 316
java.sql.ResultSet API, 300–301, 304, 317, 326–328, 343
java.sql.ResultSetMetaData API, 343
java.sql.Savepoint API, 347
java.sql.SQLException APIs, 304
java.sql.Statement API, 300, 304, 320–321, 347
java.text.CollationKey API, 400
java.text.Collator API, 399
java.text.Format API, 402
java.text.MessageFormat API, 401–402
java.text Normalizer API, 400
java.text.NumberFormat API, 383
java.text.SimpleDateFormat API, 392
java.time.format.DateTimeFormatter API, 393
java.time.LocalDate APIs, 393
java.time.ZonedDateTime API, 393
java.util.Arrays API, 8
java.util.Collection API, 5, 46
java.util.Currency API, 385
java.util.function.Supplier API, 8
java.util.Optional API, 15–16, 19
java.util.Optional APIs, 41
java.util.Random API, 40
java.util.regex.Matcher API, 140–141
java.util.regex.Pattern API, 8, 139–140
java.util.ResourceBundle API, 412–413
java.util.stream API, 35
java.util.stream.BaseStream API, 20, 45
JDBC/ODBC bridge, not available in Java 8, 283
JdbcRowSet interface, 329
JDesktopPane class, 742
getAllFrames method, 745–748, 757
no built-in cascading/tiling in, 744–748
setDragMode method, 751, 757
JDialog class, 750
JDK (Java Development Kit)
Apache Derby database, 291
DOM parser, 150
keytool program, 553
native2ascii program, 408
policytool program, 522
serialver program, 96
src.jar file, 981
StylePad demo, 682
SunJCE ciphers, 568
jdouble type (C), 947
jdoubleArray type (C), 970
JEditorPane class, 681, 712–719
addHyperlinkListener method, 718
drag-and-drop behavior of, 905
setEditable method, 714
setPage method, 714, 718
JFileChooser class
drag-and-drop behavior of, 905
setDragEnabled method, 909
jfloat type (C), 947
jfloatArray type (C), 970
JFormattedTextField class, 685–703
commitEdit method, 688, 701
constructor, 701
drag-and-drop behavior of, 905
get/setFocusLostBehavior methods, 688, 701
get/setValue methods, 687–688, 701
isEditValid method, 687, 690, 701
setText method, 688
supported formats, 691–693
JFrame class, 742
adding transfer handler to, 912
setTransferHandler method, 919
jint type (C), 947
jintArray type (C), 970
JInternalFrame class, 748–750
constructor, 742, 757
get/setContentPane methods, 758
get/setFrameIcon methods, 743, 759
getDesktopPane method, 759
is/setClosable methods, 758
is/setClosed methods, 748, 758
is/setIcon methods, 745, 758
is/setIconifiable methods, 758
is/setMaximizable methods, 758
is/setMaximum methods, 745, 758
is/setResizable methods, 758
is/setSelected methods, 744, 747, 758
is/setVisible methods, 744, 759
moveToXxx methods, 758
no built-in cascading/tiling in, 744–748
reshape method, 743, 758
show method, 759
JLabel class
extending, 661–662
using with JList, 597
JLayer class, 760–764
constructor, 763
setLayerEventMask method, 764
JList class, 582–588
addListSelectionListener method, 587
as generic type, 582
configuring for custom renderers, 597
constructor, 587, 593
drag-and-drop behavior of, 905, 913–914
get/setLayoutOrientation methods, 587
get/setPrototypeCellValue methods, 593
get/setSelectionMode methods, 583, 587
get/setVisibleRowCount methods, 583, 587
setBackground, getForeground, getSelectionXxx methods, 596, 599
getModel method, 595
getSelectedValue method, 584, 587
getSelectedValuesList method, 585, 587
HORIZONTAL_WRAP, VERTICAL, VERTICAL_WRAP values, 583
setCellRenderer method, 597, 599
setDragEnabled method, 909
setDropMode method, 913, 919
setFixedCellXXX methods, 593
visual appearance of data vs. data storage in, 588
JList.DropLocation class
getIndex method, 914, 920
isInsert method, 915, 920
jlong type (C), 947
jlongArray type (C), 970
JNDI service, 349
class loaders in, 495
JndiLoginModule class, 532
JNI (Java Native Interface), 940–1001
array elements in, 970–974
calling convention in, 950
debugging mode of, 980
error handling in, 974–979
functions in C++, 950
invoking Java methods in, 963–970
online documentation for, 952
JNI_CreateJavaVM function (C), 980–981, 985
JNI_OnLoad, JNI_OnUnload methods (C), 947
jni.h file, 947
JNICALL, JNIEXPORT macros, 942
JobAttributes class (obsolete), 886
jobject type (C), 965, 971, 988
jobjectArray type (C), 970
Join styles, 788–789
joining method (Collectors), 20, 23
JoinRowSet interface, 329
JOptionPane class, showInternal XxxDialog methods, 750
JPanel class, 857
adding layers to, 760
JPasswordField class, 681
drag-and-drop behavior of, 905
JPEG format, 823
image manipulations on, 846
printing, 874
reading, 824
transparency in, 921
JProgressBar class, 719–722, 923
constructor, 730
get/setMaximum, get/setMinimum methods, 719, 730
get/getString methods, 720, 730
getValue method, 730
is/setIndeterminate methods, 720, 730
is/getStringPainted methods, 720, 730
setValue method, 719, 730
js.properties file, 438
JScrollPane class
with JList, 582, 590
with JTabbedPane, 736
with JTable, 601
JSF (JavaServer Faces), 268
jshort type (C), 947
jshortArray type (C), 970
JSP (JavaServer Pages), 449–454
JSpinner class, 703–712
constructor, 710
getValue method, 704, 710
setEditor, setValue methods, 710
JSplitPane class, 732–735
constructor, 734
HORIZONTAL_SPLIT, VERTICAL_SPLIT values, 732
is/setContinuousLayout methods, 733, 735
is/setOneTouchExpandable methods, 732, 734
setXxxComponent methods, 735
JSSE (Java Secure Socket Extension), 579
jstring type (C), 950–951, 965, 988
JTabbedPane class, 735–741
addChangeListener method, 737, 741
addTab method, 736, 740
constructor, 736, 740
get/setComponentAt methods, 740
get/setIconAt methods, 740
get/setMnemonicAt methods, 737, 741
get/setSelectedIndex methods, 736–737, 740
get/setTitleAt methods, 740
getSelectedComponent method, 740
getTabComponentAt, getTabLayoutPolicy methods, 740
getTabCount method, 740
indexOfComponent, indexOfTab methods, 737, 740
indexOfTabComponent method, 741
insertTab, removeTabAt methods, 736, 740
setTabXxx methods, 737, 741
JTable class, 599–638
addColumn method, 617, 622
asymmetric, 608
cell editors, automatically installed, 629
constructor, 604
convertXxxIndexToModel methods, 614, 623
default rendering actions, 609
drag-and-drop behavior of, 905, 913–914
get/setAutoCreateRowSorter methods, 602, 604, 614
get/setCellSelectionEnabled methods, 613, 622
get/setColumnSelectionAllowed methods, 613, 622
get/setDefaultRenderer methods, 626–627, 637
get/setFillViewportHeight methods, 602, 604
getColumnModel method, 622
getDefaultEditor method, 637
getRowXxx methods, 622
getSelectionModel method, 622
moveColumn, removeColumn methods, 617, 623
not generic, 601
print method, 602, 604
resize modes, 611
setAutoResizeMode method, 611, 622
setDragEnabled method, 909
setDropMode method, 913, 919
JTable class (continued)
  setRowHeight, setRowMargin, setRowSelectionAllowed
  methods, 612, 622
setRowSorter method, 614, 623
JTextArea class, 681
  drag-and-drop behavior of, 905
extending, 524
JTextComponent class
  drag-and-drop behavior of, 913–914
  setDragEnabled method, 909
  setDropMode method, 913, 919
JTextField class, 681
  drag-and-drop behavior of, 905
  with DefaultCellEditor, 629
JTextPane class, 681
  drag-and-drop behavior of, 905
JUnit 4 tool, 456
Just-in-time compiler, 980
JVM (Java virtual machine)
  bootstrap class loader in, 493
class files in, 492
  creating, 980
  embedding into native code, 980–985
  passing command-line arguments to, 515
  terminating, 509–530, 981
  transferring object references in, 902
K
\k, in regular expressions, 131
Kerberos protocol, 531, 579
kernel class, 845, 851
Kernel, of a convolution, 844
KEY_Xxx rendering hints, 817–818
Key/value pairs. See Properties
Keyboard, reading from, 48, 60
KeyGenerator class, 569
  generateKey method, 569, 574
  getInstance method, 574
init method, 574
KeyPairGenerator class, 576
Keys
  autogenerated, 320–321
  generating, 569–574
  primary, 320
  keystore keyword, 563
  KeyStoreLoginModule class, 532
Keystores, 553–556, 563
  referencing in policy files, 563
Keystrokes
  filtering, 685
  monitoring, 682
  keytool program, 553–556
  Krb5LoginModule class, 532
L
l
  (object), type code, 87, 961
in masks, 692
Lambda expressions, using with map method, 9
Landscape orientation, 802
Language codes, 374
Language Model API, 476–477
Language tags, 377
last (ResultSet), 323, 327
lastInMonth method (TemporalAdjusters), 359
lastXxxTime methods (BasicFileAttributes), 109
layer/ColorFrame.java, 761
Layers (Swing), 760–764
LayerUI class, 760–764
  extending, 760
  installUI, uninstallUI methods, 764
  paint method, 764
  processXxxEvent methods, 761, 764
Layout algorithm, 862–863
layoutPages method (Banner), 863
LCD displays, 836
LD_LIBRARY_PATH environment variable, 946, 985
LDAP (Lightweight Directory Access Protocol), 579
Leaf icons, 661
Leap seconds, 352
Leap years, 356
Learn SQL The Hard Way (Shaw), 285
Learning SQL (Beaulieu), 285
Leaves (Swing), 639, 647, 672
icons for, 648, 661
Legacy data, converting into XML, 227
length method
of Blob, Clob, 317
of CharSequence, 55
of RandomAccessFile, 73, 76
LIB environment variable, 984
Libraries, replacing with newer versions, 494
LIKE statement (SQL), 289, 319
limit method (Stream), 10–11, 43, 122
Line feed, 61, 405
in e-mails, 278
in regular expressions, 132
Line2D class, 769
lines method (Files), 6, 8
Lines, reading, 6
lineTo method (Path2D.Float), 775, 786
Linux operating system
compiling invocation API, 984
Java deployment directory in, 564
library path in, 946
OpenSSL in, 560
using GNU C compiler, 943
list method (Files), 110
list boxes (Swing), 582–588
constructing, 582
from arrays/vectors, 594–595
new components for each cell, 597
double-clicking in, 585
fixed cell sizes in, 590
layout orientation of, 583
scrolling, 582, 590
selecting multiple items in, 583, 585
using with split panes, 733
values of:
computing when called, 589
editing, 593–595
rendering, 595–599
very long, 589–590
List cell renderers, 595–599
list interface, 704
List models (Swing), 588–593
List selection events, 584
list/ListFrame.java, 585
ListCellRenderer interface, 595, 597
getListCellRendererComponent method, 595–599
ListDataListener interface, 589
ListModel interface
getElementAt, getSize methods, 589, 593
implementing, 588–589, 594
listRendering/FontCellRenderer.java, 598
ListResourceBundle class, 411
ListSelectionEvent class, 584
gValueIsAdjusting method, 584
ListSelectionListener interface, valueChanged method, 584, 588
ListModel interface
selection modes, 612
setSelectionMode method, 612, 624
LITTLE_ENDIAN constant (ByteOrder), 123
Little-endian order, 70, 118, 407
Load time, 486
loadClass method
of ClassLoader, 496, 498
of URLClassLoader, 494
loadLibrary method (System), 944, 946
LOBs (large objects), 316–318
creating empty, 318
placing in database, 316
reading, 316
Local hosts, 240
Local names, 198
Local variables, annotating, 466
Locale class
and legacy classes, 369
methods of, 356–357, 386, 393
localdates/LocalDates.java, 358
LocalDateTime class, 361
and legacy classes, 369
atZone method, 361
methods of, 386, 393
Locale, 373–377
constructor, 377
debugging, 377
eforLanguageTag method, 377
get/setDefault methods, 376–377
getCountry method, 377
getDisplayCountry, getDisplayLanguage methods, 377
getDisplayName method, 376–377, 379
getInstance method, 393
getLanguage method, 377
toLanguageTag method, 375, 377
toString method, 377
Locales, 28, 372–377
  and resources bundles, 409–410
  current, 401
default, 365, 376
display names of, 376
formatting styles for, 366, 386–387
numbers in, 379, 686
predefined, 375
supported by classes, 376
variants in, 373, 410
LocalTime class, 360–361
  and legacy classes, 369
  methods of, 360, 386, 393
lock method (FileChannel), 126–128
Locks, 126–128
  for the tail portion of a file, 127
  shared, 127
  unlocking, 127
Log files, 406
Log messages, adding to classes, 481–486
@LogEntry annotation, 481
Logging, code generation for, 455
LoggingPermission class, 520
LoginContext class, 531
  constructor, 536
  getSubject method, 536
  login, logout methods, 531, 536
LoginException, 536
LoginModule interface, methods of, 546
Logins
  committed, 540
  modules for, 532
    custom, 537–546
    separating from action code, 539
Long class, MAX_VALUE constant, 127
LONG VARCHAR data type (SQL), 348
long type
  printing, 61
  type code for, 87, 961
  vs. C types, 947
  writing in binary format, 69
LONG VARCHAR data type (SQL), 348
LongBuffer class, 124
LongList/LongListFrame.java, 591
LongList/WordListModel.java, 592
longs method (Random), 37, 40
LongStream interface, 36–41
  average, max, min, sum methods, 37, 39
  boxed method, 36, 39
mapToLong method, 36
  of method, 39
  range, rangeClosed methods, 36, 39
  summaryStatistics method, 37, 39
toArray method, 36, 39
LongSummaryStatistics class, 20, 23, 37, 41
Look-and-feel
displaying trees in, 645
  handles for subtrees in, 661
  hot keys in, 686
  selecting multiple nodes in, 665
lookingAt method (Matcher), 140
Lookup tables, 411
LookupTable class, 843–844
  constructor, 850
lookupPrintServices method (PrintServiceLookup), 874, 877
lookupStreamPrintServiceFactories method (StreamPrintServiceFactory), 879
LookupTable class, 844
lostOwnership method (ClipboardOwner), 888, 892
LSB (least significant byte), 70
LSOutput interface, 211
LSSerializer interface, 210
&amp;, entity reference, 148

M
Mac OS X
  character encodings in, 404
  OpenSSL in, 560
  resources in, 408
mail method (Desktop), 932
Mail messages/headers, 277–280
mail/MailTest.java, 279
mailto: URI scheme, 927
main method
executing, 493
  setting security managers in, 516
makeShape method (ShapeMaker), 776–777
makeVisible method (JTree), 653, 657
Mandelbrot set, 837–838
Mangling names, 943, 961
Manifest files, 77, 921
map method
  of FileChannel, 117, 122
  of Optional, 14–15
  of Stream, 9–10
Map interface, 817
mapping method (Collectors), 30, 33
Maps
  concurrent, 25
  merging, 24–27, 43
mapToXxx methods (XxxStream), 36
mark method
  of Buffer, 125–126
  of InputStream, 50
Marker annotations, 465
markSupported method (InputStream), 50
MaskFormatter class, 692–693
  constructor, 703
  get/setInvalidCharacters methods, 692, 703
  get/setPlaceholder methods, 703
  get/setValidCharacters methods, 692, 703
  get/setValueContainsLiteralCharacters methods, 703
  getPlaceholderCharacter method, 703
  overtype mode, 693
  setPlaceholderCharacter method, 693, 703
Mastering Regular Expressions (Friedl), 134
match attribute (XSLT), 224
match/HrefMatch.java, 137
matcher method (Pattern), 134, 140
Matcher class
  end method, 135, 137, 140–141
  find method, 137, 140
  group method, 135, 140–141
  groupCount method, 141
  lookingAt method, 140
  matches method, 134
  quoteReplacement method, 141
  replaceXxx methods, 139, 141
  reset method, 141
  start method, 135, 137, 140–141
  matches method (Matcher), 134
Matrices, transformations of, 802
max method (of streams), 12, 37, 39–40
MAX_VALUE constant (Long), 127
maxBy method (Collectors), 30, 33
maxOccurs attribute (XML Schema), 174
MD5 algorithm, 548–549
MDI (multiple document interface), 741–742
Memory addresses, vs. serial numbers, 84
Memory mapping, 116–128
  modes of, 117
memoryMap/MemoryMapTest.java, 119
Message digests, 547–550
Message signing, 550–553
MessageDigest class
  digest method, 549–550
  extending, 548
  getInstance method, 548–550
  reset method, 550
  update method, 549–550
MessageFormat class, 400–404, 414
  applyPattern method, 401
  constructor, 401
  format method, 401–402
  get/setLocale methods, 402
  ignoring the first limit, 404
Messages, formatting, 400–404
Meta-annotations, 456, 472–475
Metadata (databases), 333–343
Metal look-and-feel
  hot keys in, 686
  internal frames in, 742–744
  one-touch expand icons in, 732
  selecting multiple nodes in, 665
  trees in, 644–645
Method class, 457
Method references, type use annotations in, 468
Method verification errors, 506, 508
Methods
  adding logging messages to, 481–486
  annotating, 456, 466, 470
  calling from native code, 963–970
  getters/setters, generated automatically, 480
  instance, 963–964
  mangling names of, 943, 961
  native, 939–1001
  of annotation interfaces, 463
  overriding, 471
  signatures of, 961–963
  static, 964–965
  vetoable, 748
Microsoft
  compiler, 944
  invocation API in, 984
  ODBC API of, 282
  SQL Server, 291
  single active statements in, 301
MIME (Multipurpose Internet Mail Extensions), 824
MIME types, 892–894
  for print services, 875
MimeMessage class, methods of, 278
min method (of streams), 12, 37, 39–40
minBy method (Collectors), 30, 33
minoccurs attribute (XML Schema), 174
minus, minusXxx methods (Date and Time API), 353, 356–357, 360, 363
MissingResourceException, 409
Miter join, 788–789
Miter limit, 790
Mixed content (XML), 147
parsing, 166
Model-view-controller design pattern, 588
Modernist painting example, 211
Modified UTF-8, 70–72, 407
and native code, 949–952
Monads, 10
Month enumeration, 355
getDisplayName method, 365, 386
MonthDay class, 357
Mouse, double-clicking on list components, 585
move method (Files), 106, 108
moveColumn method (JTable), 617, 623
moveTo method (Path2D.Float), 775–776, 786
moveToXxx methods (JInternalFrame), 775–776, 786
moveToXxxRow methods (ResultSet), 325, 327
MSB (most significant byte), 70
Multiple-page printing, 861–864
multipliedBy method (Instant, Duration), 353
MutableTreeNode interface
implementing, 641
setUserObject method, 642, 649
MySQL database, 291

N
\n
as line feed, 61, 154, 405
in e-mails, 278
in regular expressions, 130–131
NameCallback class, 538
constructor, 545
methods of, 545
NamedNodeMap interface, getLength, item methods, 162
names method (Win32RegKey), 987
Namespaces, 196–199
activating processing of, 174
aliases (prefixes) for, 172, 197
of attributes, 198
of child elements, 197
using class loaders as, 496
Nashorn engine, 430–431
National character strings, 348
National Institute of Standards and Technology, 234, 548
native keyword, 940
Native applications
communicating with Java, 903–921
data transfers between Java and, 888
Native methods, 939–1001
and garbage collection, 951
array elements in, 970–974
class references in, 958
compiling, 943
enumerating keys with, 989
error handling in, 974–979
exceptions in, 975
instance fields in, 956–960
invoking Java constructors in, 965–966
linking to Java, 946
naming, 941–942
overloading, 941
reasons to use, 940
registry access functions in, 988–1001
static, 941
static fields in, 960–961
strings in, 949
native2ascii program, 408, 410, 414
NCHAR data type (SQL), 348
NCHAR, NCLOB data types (SQL), 348
negated method (Instant, Duration), 353
Negative byte values, 693
Nervous text applet, 546
NetBeans, startup plugins for, 922
NetPermission class, 519
Networking, 233–280
blocking worker threads indefinitely, 723
debugging, 233–236
getting huge images from, 826
newBufferedXxx methods (Files), 104
newDirectByteBuffer function (C), 973
newDirectoryStream method (Files), 111, 114
newDocument method (DocumentBuilder), 209, 212, 227
newDocumentBuilder method (DocumentBuilderFactory), 150, 160, 209
newFactory method (RowSetProvider), 329, 332
newFileSystem method (FileSystems), 115–116
NewGlobalRef function (C), 958
newInputStream method
of Channels, 256
of Files, 104–105
newInstance method
  of DocumentBuilderFactory, 150, 160
do SAXParserFactory, 201, 203
do TransformerFactory, 213
do XMLInputFactory, 207
do XMLStreamFactory, 214, 221
do XPathFactory, 191, 195
NewObject function (C), 965, 970, 974
NewOutputStream method
  of Channels, 250, 257
  of Files, 104–105
newPath method (XPathFactory), 195
newSAXParser method (SAXParserFactory), 201, 203
NewString function (C), 953
NewStringUTF function (C), 950–952, 954, 988
newTransformer method (TransformerFactory), 213, 231
NewXxxArray functions (C), 973, 988
next method
  of ResultSet, 298, 300, 321
  of XMLStreamReader, 208
next, nextOrSame methods (TemporalAdjusters), 359
nextElement method (Enumeration), 659, 987–990
nextPage method (CachedRowSet), 330, 332
NMTOKEN, NMTOKENS attribute types (DTDs), 167–168
Node interface
  appendChild method, 209, 212
  getAttributes method, 154, 161
  getFirstChildNodes method, 151, 161
  getFirstChild method, 153, 161
  getLastChild method, 154, 161
  getLocalName method, 198–199
  getNamespaceURI method, 198–199
  getNextSibling method, 154, 161
  getNodeXxx methods, 154, 161, 198
  getParentNode method, 161
  getPreviousSibling method, 161
  subinterfaces of, 151
Node renderer, 648
nodeChanged method (DefaultTreeModel), 652, 658
Nondeterministic parsing, 166
noneMatch method (Stream), 13
Noninterference, of stream operations, 43
NonNull annotation, 467–468
normalize method (Path), 102–103
Normalized color values, 835
Normalizer class, 395
  normalize method, 400
NoSuchAlgorithmException, 550, 573
NoSuchElementException, 15, 990
notFilter method (RowFilter), 615, 625
NotSerializableException, 92
now method (Date and Time API), 352, 356, 360, 363
NT logins, 531
NTLoginModule class, 532–533
NTUserPrincipal class, 533
NullPointerException, 12, 976
Number class
  doubleValue method, 378
  intValue method, 379
numberFilter method (RowFilter), 615, 625
NumberFormat class, 378–383, 691
  format method, 379, 383
  get/setXxxDigits methods, 383
  getAvailableLocales method, 376, 379, 383
  getCurrencyInstance method, 378, 383–384, 691
  getIntegerInstance method, 689
  getNumberInstance method, 378, 383, 691
  getPercentInstance method, 378, 383, 691
  is/setGroupingUsed methods, 383
currently selected, 650
ingoing, 654, 673
enumerating, 659–661
expanding, 654
handles for, 644, 646, 661
highlighting, 662
identifying, by tree paths, 650
making visible, 653
parent, 639, 642
rendering, 661–664
root, 639–647
row positions of, 652
searching, for a given user object, 660, 666
selecting, 665
user objects for, 642
changing, 652
nodesChanged method (DefaultTreeModel), 658
Nondeterministic parsing, 166
lessMatch method (Stream), 13
Noninterference, of stream operations, 43
NonNull annotation, 467–468
normalize method (Path), 102–103
Normalized color values, 835
Normalizer class, 395
  normalize method, 400
NoSuchAlgorithmException, 550, 573
NoSuchElementException, 15, 990
notFilter method (RowFilter), 615, 625
NotSerializableException, 92
now method (Date and Time API), 352, 356, 360, 363
NT logins, 531
NTLoginModule class, 532–533
NTUserPrincipal class, 533
NullPointerException, 12, 976
Number class
  doubleValue method, 378
  intValue method, 379
numberFilter method (RowFilter), 615, 625
NumberFormat class, 378–383, 691
  format method, 379, 383
  get/setXxxDigits methods, 383
  getAvailableLocales method, 376, 379, 383
  getCurrencyInstance method, 378, 383–384, 691
  getIntegerInstance method, 689
  getNumberInstance method, 378, 383, 691
  getPercentInstance method, 378, 383, 691
  is/setGroupingUsed methods, 383
NumberFormat class (continued)
   is/setParseIntegerOnly methods, 383
   parse method, 378–379, 383
   setCurrency method, 384
   supported locales, 376
   numberFormat/NumberFormatTest.java, 380
   NumberFormatException, 683
Numbers
   filtering, 615
   floating-point, 372, 378–383, 691
   formatting, 372, 378–383, 691
      supported locales for, 379
         with C, 947
   in regular expressions, 131, 133
   printing, 61
   random, 37
   reading:
      from files, 56
      from ZIP archives, 58
      using locales, 378
   truly random, 570
   writing in binary format, 69
NUMERIC data type (SQL), 291, 348
NVARCHAR data type (SQL), 348

0
Object class, clone method, 81, 98
Object inspection tree, 671–680
Object references, and clipboard, 902
Object serialization, 80–100
   cloning with, 98–100
   file format for, 85–92
   MIME type for objects in, 893
   modifying default, 92–94
   of singletons, 94–95
   serial numbers for, 82–83
ObjectInputStream class, 81
   constructor, 85
   readObject method, 81, 85, 93
ObjectOutputStream class, 80
   constructor, 85
   defaultWriteObject method, 92
   writeObject method, 80, 85, 92
Objects
   cloning, 98–100
   fingerprints of, 87
   life cycle of, 472
   MIME types for, 893
   printing, 61
   reading from an input stream, 81
saving:
   in database, 474
   in output streams, 80, 82
   in text format, 63–67
   serializable, 80–85
   transferring via clipboard, 898–902
   transmitting over network connection, 84
   type code for, 87, 961
   versioning, 95–98
objectStream/ObjectStreamTest.java, 84
ODBC API, 282, 284
of method
   of Date and Time API, 356, 360–363
   of DoubleStream, 39
   of IntStream, 36, 38
   of LongStream, 39
   of Optional, 16
   of Stream, 5, 8
ofDateAdjuster method (TemporalAdjusters), 359
offsetDateTime class, 364
ofInstant method (ZonedDateTime), 363
ofLocalizedXxx methods (DateTimeFormatter), 365,
   385, 392
ofNullable method (Optional), 16
ofPattern method (DateTimeFormatter), 367
oj literal (SQL), 319
One-touch expand icon, 732
open method
   of Desktop, 927, 932
   of FileChannel, 117, 122
   of SocketChannel, 250, 256
openConnection method (URL), 259, 265
Opened nonleaf icons, 648, 661
openOutputStream method (SimpleJavaFileObject), 445,
   449
OpenSSL toolkit, 560–561
openStream method (URL), 257, 265
Operating system
   character encodings in, 404
   resources in, 408
Optional class, 12–17
   creating values of, 16
   empty method, 16
   flatMap method, 16–19
   get method, 15–16
   ifPresent method, 14–15, 41
   isPresent method, 15–16
   map method, 14–15
   of, ofNullable methods, 16
   orElse method, 12, 14–15, 41
orElseGet method, 14–15, 41
orElseThrow method, 15
optional keyword, 532
orElse/OptionalTest.java, 17
OptionalXXX classes, 37, 41
Oracle JVM implementation, 973
order method (ByteBuffer), 118, 123
ORDER BY statement (SQL), 299
orElse method (Optional), 12, 14–15, 41
orElseGet method (Optional), 14–15, 41
orElseThrow method (Optional), 15
orFilter method (RowFilter), 615, 625
org.w3c.dom package, 150
org.w3c.dom.CharacterData API, 162
org.w3c.dom.Document API, 161, 212
org.w3c.dom.Element API, 161, 212
org.w3c.dom.Node API, 161, 199, 212
org.w3c.dom.NodeList API, 162
org.w3c.dom.NamedNodeMap API, 162
org.xml.sax.Attributes API, 204–205
org.xml.sax.ContentHandler API, 204
org.xml.sax.EntityResolver API, 170
org.xml.sax.ErrorHandler API, 171
org.xml.sax.helpers.AttributesImpl API, 232
org.xml.sax.InputSource API, 171
org.xml.sax.SAXParseException API, 171
org.xml.sax.XMLReader API, 232
Outer joins, 319
Outline dragging, 751–760
OutOfMemoryError, 976
output element (XSLT), 224
Output streams, 48–59
and Unicode, 48
buffered, 57–59
byte processing in, 57
byte-oriented, 48
closing, 49, 249
filters for, 55–59
hierarchy of, 51–55
objects in, 80–100
OutputStream class, 48, 52–53, 214
close method, 51
flush method, 49, 51
write method, 48, 50–51
OutputStreamWriter class, 60
OverlappingFileLockException, 127
Overloading, 941
@Override annotation, 470–471
Overtype mode, 693
Overwrite mode, 692

P
\p, \P, in regular expressions, 130
Package class, implementing AnnotatedElement, 457
package-info.java file, 467
Packages
annotating, 466–467, 470
avoiding name clashes with, 196, 496
Packets, 238
Padding schemes, 568
Page setup dialog box, 856–857
Pageable interface
implementing, 862
objects, printing, 874
PageAttributes class (obsolete), 886
pageDialog method (PrinterJob), 856–857, 861
PageFormat class
getHeight, getWidth methods, 854, 861
getImageableXXX methods, 855, 861–862
getOrientation method, 862
Pages
measurements of, 855
multiple, printing, 862–864
orientation of, 802, 856, 862
paint method
of JComponent, 626, 766
of LayerUI, 764
of PanelLayer, 760
Paint, 797–799
Paint interface, 797
paintComponent method
of JComponent, 596–597, 766
of StrokePanel, 791
Paper margins, 854
Paper sizes, 412, 855
parallel method (BaseStream), 41–45
parallel/ParallelStreams.java, 44
parallelStream method (Collection), 2–5, 41–46
Parameter variables, annotating, 466
Parent nodes (Swing), 639, 642
parse method
of DateTimeFormatter, 367
of DocumentBuilder, 160
of LocalXXX, ZonedDateTime, 386, 393
of NumberFormat, 378–379, 383
of SAXParser, 201, 203
of XMLReader, 232
Parsed character data, 165
ParseException, 379, 383, 693
Parsers, 149–162
  checking uniqueness of IDs in, 168, 178
  pull, 205
  validating in, 163
Parsing (XML), 149–162
  nondeterministic, 166
  with XML Schema, 174
partitioningBy method (Collectors), 28, 31
PasswordCallback class, 538
  constructor, 546
  getPassword methods, 546
  getPrompt method, 546
  isEchoOn method, 546
Password-protected resources, 261–262
Path interface, 101–103
  getParent, getFileName, getRoot methods, 103
  normalize, relativize methods, 102–103
  resolve, resolveSibling methods, 102–103
  toAbsolutePath, toFile methods, 102–103
  toRelative, toRelative method, 102–103
Path2D class, append, closePath methods, 776, 786
Path2D.Float class
  curveTo, lineTo, quadTo methods, 775, 786
  moveTo method, 775–776, 786
pathFromAncestorEnumeration method
  (DefaultMutableTreeNode), 660
Paths (file system), 101–103
  absolute vs. relative, 56, 101–102
  checking properties of, 108–109
  filtering, 111
  relativizing, 102
  resolving, 58, 101
  root component of, 101
  separators in, 56, 101
Paths (graphics)
  closing, 776
  drawing, 775–776
Paths class, 115
  get method, 101, 103
Pattern class, 134
  compile method, 134, 139
  matcher method, 134, 140
  split method, 139–140
  splitAsStream method, 6, 8
Patterns, 128–141
#PCDATA element content (DTD), 165
PDF format, printing, 874
peek method (Stream), 11–12
Pentium processor, little-endian order in, 70
Percentages, formatting, 378–383, 691
Performance
  of encryption algorithms, 576
  of file operations, 116–124
Period class, 357
Perl programming language, regular
  expressions in, 134
Permission class
  constructor, 530
  equals method, 523
  extending, 522
  getActions method, 523
  getName method, 525, 530
  hashCode method, 523
  implies method, 523–524, 530
Permission files, 511
permission keyword, 517, 520
Permissions, 510–514
  call stack of, 513
  checking, 509–510
  class hierarchy of, 512
  commonly used classes for, 517–520
  for files, 520
  for users, 531
  implementing, 522–530
  implying other permissions, 524
  in policy files, 514–522
  mapping code sources to, 510
  order of, 523
  property, 521
  restricting to certain users, 533
  socket, 521
  targets of, 520–521
permissions/PermissionTest.java, 528
permissions/WordCheckPermission.java, 526
Permutations, of a string, 705–706
@Persistent annotation, 474
Personal data, transferring, 567
Pixels
  affine transformations on, 843
  antialiasing, 818–819
  average value of, 844
  composing, 807–817
  interpolating, 797–798, 818, 820, 843
  reading, 835
  setting individual, 835–842
Placeholder characters, in masks, 693
Placeholders, in message formatting, 400–404
PlainDocument class, 681, 689
Plugins, loading, 494
pluses, plusXXX methods (Date and Time API), 353, 356–357, 360, 362–363
PNG format, 823
animated, 921
printing, 874
Points, in typography, 855
Policy class, 511, 515
Policy files, 514–522
and Java Plug-in, 563
building, 563–565
debugging, 564
locations for, 515
parsing, 525
platform-independent, 521
referencing keystores in, 563
system properties in, 521
user roles in, 530–546
visibility of, 525
Policy managers, 514–522
policytool program, 522
Polygons, 769, 776
populate method (CachedRowSet), 330, 332
Pop-up menus, for tray icons, 933
PopupMenu class, 933
Porter-Duff rules, 808–811
Portrait orientation, 856
Ports, 235
blocking, 234
in URIs, 258
position method (Buffer), 126
position function (XPath), 225
POSIX-compliant file systems, 109
PosixFileAttributes interface, 109
POST method (HTML), 269–271
post/PostTest.java, 274
@PostConstruct annotation, 470, 472
PostgreSQL database, 291
connecting to, 295
drivers for, 292
Postorder traversal, 659
postOrderEnumeration method (DefaultMutableTreeNode), 659, 663
PostScript format
printing, 874, 878
writing to, 878
Predefined character class names, 129, 133
Predefined character classes, 130
@PreDestroy annotation, 470, 472
premain method (Instrumentation API), 487
preOrderEnumeration method (DefaultMutableTreeNode), 659, 663
Prepared statements, 309–316
caching, 311
executing, 310
PreparedStatement interface
clearParameters method, 316
executeXxx, setXxx methods, 310, 316
prepareStatement method (Connection), 310, 315, 322, 326
previous method (Resultset), 323, 327
previous, previousOrSame methods (TemporalAdjusters), 359
previousPage method (CachedRowSet), 332
preVisitDirectory, postVisitDirectory methods
of FileVisitor, 112
of SimpleFileVisitor, 115
Primary keys, 320
Primitive types
arrays of, 973
I/O in binary format in, 51
streams of, 36–41
Principal interface, getName method, 537
Principals (logins), 532
print method
of Desktop, 927, 932
of DocPrintJob, 877
of JTable, 602, 604
Print dialog box, 852
displaying page ranges in, 854, 862
native, 853, 857
Print preview, 864–873
Print services, 874–878
document flavors for, 874–875
for images, 876
stream, 878–879
print/PrintComponent.java, 859
print/PrintTestFrame.java, 858
Printable interface
implementing, 852, 857
objects, printing, 874
print method, 852, 860, 862, 864–865
Printer graphics context, 863
PrinterException, 853
PrinterJob class
defaultPage method, 861
getPrinterJob method, 852, 861
pageDialog method, 856–857, 861
print method, 853–854, 861
printDialog method, 852–853, 861
PrinterJob class (continued)
   setPageable method, 862, 873
   setPrintable method, 861
printf function (C), 947
printf1/Printf1.c, 948
printf1/Printf1.java, 948
printf1/Printf1Test.java, 949
printf2/Printf2.c, 954
printf2/Printf2.java, 954
printf3/Printf3.c, 967
printf3/Printf3.java, 967
printf3/Printf3Test.java, 966
printf4/Printf4.c, 976
printf4/Printf4.java, 978
printf4/Printf4Test.java, 979
Printing, 851–887
   clipped areas, 854
   counting pages during, 854
   images, 852–862
   layout of, 862–863
   multipage documents, 861–864
   number of copies for, 879
   page orientation of, 802, 856
   paper sizes in, 855
   quality of, 882
   selecting settings for, 852
   starting, 509, 852
   text, 852–862
   using:
      banding for, 854
      transformations for, 864
Printing attributes, 879–887
   adding, 881
   categories of, 881–882
   checking values of, 882
   hierarchy of, 880
   retrieving, 882
PrintJob class (obsolete), 852
PrintJobAttribute interface, 879
   printing attributes of, 882–885
PrintJobAttributeSet interface, 880
println method (System.out), 405–406
PrintPreviewCanvas class, 864
PrintPreviewDialog class, 864–873
PrintQuality class, 882
PrintRequestAttribute interface, 879
   printing attributes of, 882–885
PrintRequestAttributeSet interface, 852, 880
PrintService interface
   createPrintJob method, 874, 877
   getAttributes method, 886
   getName method, 874
PrintService/PrintServiceTest.java, 876
PrintServiceAttribute interface, 879
   printing attributes of, 882–885
PrintServiceAttributeSet interface, 880
PrintServiceLookup class, lookupPrintServices method, 874, 877
PrintStream class, 61
PrintWriter class, 56, 60–62
   checkError method, 61–62
   constructor, 62
   print method, 61–62, 963–964
   println methods, 61–62
Privacy Enhanced Mail (PEM) format, 560
Private keys, 550–563, 576
PrivilegedAction interface, 539
   implementing, 532
   run method, 532, 536
PrivilegedExceptionAction interface, 533
   run method, 537
processAnnotations method (ActionListenerInstaller), 457–458
Processing instructions (XML), 149
Processing tools, 455
Processor interface, 476
processXxxEvent methods (LayerUI), 761, 764
Programmer’s Day, 356
Programs. See Applications
Progress bars (Swing), 719–722
   completion percentage in, 720
   indeterminate, 720
   minimum/maximum values of, 719
   on a splash screen, 922–923
Progress monitors (Swing), 722–726
   cancellation requests in, 723
   for input streams, 726–731
   timers in, 723
progressBar/ProgressBarFrame.java, 720
ProgressMonitor class, 722–726
   close method, 723, 731
   constructor, 731
   isCanceled method, 723, 731
   setMillisToXxx methods, 724
   setNote method, 731
   setProgress method, 723, 731
progressMonitor/ProgressMonitorFrame.java, 724
ProgressMonitorInputStream class, 726–731
  constructor, 731
  read method, 727
progressMonitorInputStream/TextFrame.java, 728
Properties
  constrained, 748–750
  generated automatically, 480
Properties class, 144
Property files, 144
  character encoding of, 410
  event handlers in, 438
  for resources bundles, 409–410
  for string resources, 408
  for strings, 410
  no passwords in, 278
Property permissions, 521
@property annotation, 480
PropertyChangeEvent class
  getNewValue method, 750, 759
  getPropertyName method, 759
PropertyPermission class, 517
PropertyVetoException class, 744–750
  constructor, 760
Protection domains, 512
ProtectionDomain class
  constructor, 514
  getCodeSource, implies methods, 514
Prototype cell values, 590
Proxy objects, 457
  for annotated methods, 458
PUBLIC identifier (DTD), 210
Public certificates, keystore for, 563
Public key ciphers, 550–558, 575–579
  performance of, 576
Public Key Cryptography Standard (PKCS) #5, 568
Pull parsers, 205
PushbackInputStream class, 57
  constructor, 59
  unread method, 59
put method
  of Bindings, 434
  of ByteBuffer, 122
  of CharBuffer, 124
  of ScriptEngine, 433
  of ScriptEngineManager, 433
putClientProperty method (JComboBox), 645, 650, 751
putNextEntry method (ZipOutputStream), 77–78
putXxx methods (ByteBuffer), 118, 123

Q
\0, in regular expressions, 130
QBE (query by example) tools, 287
QuadCurve2D class, 769, 774
QuadCurve2D.Double class, 785
Quadratic curves, 774–775
quadTo method (Path2D.Float), 775, 786
Qualified names, 198
Quantifiers, 129
Queries (databases), 288–290
  by example, 287
  executing, 298, 309–321
  multiple, 301
  populating row sets with results of, 330
  preparing, 309–316
  returning multiple results, 319–320
query\QueryTest.java, 312
&quot;;, entity reference, 148
quoteReplacement method (Watcher), 141

R
R programming language, 430, 438
\r line feed character, 61, 154, 405
  in e-mails, 278
\r, \r, in regular expressions, 130, 132
Race conditions, 42
Random class, 570
  methods of, 37, 40
Random numbers, 37
Random-access files, 72–76
randomAccess/RandomAccessTest.java, 74
RandomAccessFile class, 72–76, 116
  constructor, 76
  getChannel method, 121
  getFilePointer method, 72, 76
  length method, 73, 76
  seek method, 72, 76
Randomness, 570
range, rangeClosed methods (XxxStream), 36, 38–39
Raster class
  getDataElements method, 837, 841
  getDataElements method, 837, 841
  getPixel, getPixels methods, 836, 841
rasterImage/RasterImageFrame.java, 839
read method
  of CipherInputStream, 575
  of FileInputStream, 48
  of ImageIO, 823, 831
  of ImageReader, 832
  of InputStream, 48–50
read method (continued)
of ProgressMonitorInputStream, 727
of ZipInputStream, 77
read/fontdialog.xml, 185
read/gridbag.dtd, 187
read/gridbag.xsd, 188
read/GridBagPane.java, 181
read/GridBagTest.java, 179
Readable interface, 52
read method, 53, 55
ReadableByteChannel interface, 250
readAllXXX methods (Files), 104
readAttributes method (Files), 109
Reader class, 48, 52
read method, 51
READER class (DocFlavor), 875
readExternal method (Externalizable), 93–94
readFixedString method (DataInput), 73–74
readline method (Console), 63
readObject method
of Date, 93
of ObjectInputStream, 81, 85, 93
ReadOnlBufferException, 117
readResolve method (Serializable), 95
readThumbnail method (ImageReader), 833
readXXX methods (DataInput), 70–71, 81
REAL data type (SQL), 291, 348
Receiver parameters, 469
Rectangle2D class, 769
RectangularShape class, 770
reduce method (Stream), 33–35
Reductions, 12, 33–35
ref attribute (XML Schema), 173
Reflection
accessing fields of another class with, 509
constructing:
class tree with, 665
static field names with, 387
enumerating fields from a variable with, 674
ReflectPermission class, 519
regex/RegexTest.java, 136
regexFilter method (RowFilter), 615, 625
Registry editor, 986, 990
Registry keys, 987–989
Regular expressions, 128–141
escapes in, 64, 129
filtering, 615
formatters for, 694
grouping in, 133, 135–136
in DTDs, 165
predefined character classes in, 129, 133
quantifiers in, 129
replacing all matches with, 139
relative method (ResultSet), 323, 327
Relative URLs, 563
Relativization, of an absolute URL, 259
relativize method (Path), 102–103
releaseSavepoint method (Connection), 345–346
ReleaseStringChars function (C), 953
ReleaseStringUTFChars function (C), 951–952, 954
releaseXxxArrayElements functions (C), 972, 974
reload method (DefaultTreeModel), 653, 658
remaining method (Buffer), 125–126
Remote objects, MIME type for, 893
remove method
of AttributeSet, 886
of DocumentFilter, 702
of SystemTray, 936
removeActionListener method (TrayIcon), 937
removeCellEditorListener method (CellEditor), 638
removeColumn method (JTable), 617, 623
removeElement method (DefaultListModel), 595
removeNodeFromParent method (DefaultTreeModel), 652, 658
removeTab method (JTabbedPane), 736, 740
removeTreeModelListener method (TreeModel), 672, 680
removeUpdate method (DocumentListener), 682, 685
Rendering (Swing)
cells, 626–638
columns, 609–610
headers, 627
lists, 595–599
nodes, 661–664
Rendering hints, 766, 817–823
Rendering pipeline, 766–768
RenderingHints class, 817, 823
renderQuality/RenderQualityTestFrame.java, 820
Renjin project, 430–431, 439
@Repeatable annotation, 471
replace method (DocumentFilter), 689, 702
replaceXXX methods (Matcher), 139, 141
required keyword, 532
#REQUIRED attribute (DTD), 167
requisite keyword, 532
RescaleOp class, 843, 850
Rescaling operation, 843
reset method
  of Buffer, 125–126
  of InputStream, 50
  of Matcher, 141
  of MessageDigest, 550
reshape method (JInternalFrame), 743, 758
resolve, resolveSibling methods (Path), 102–103
resolveEntity method (EntityResolver), 164, 170
Resolving
  classes, 492
  relative URLs, 259
Resource bundles, 408–413
  loading, 410–411
  locating, 409–410
  lookup tables for, 411
  naming, 411
  searching for, 411
Resource editor, 408
@Resource annotation, 349, 470, 472
ResourceBundle
  class extending, 411–412
  getBundle method, 409–412
  getKeys method, 412–413
  getObjectId method, 411–412
  getString method, 410, 412
  getStringsArray method, 412
  handleGetObject method, 412–413
ResourceBundle(Control class, getCandidateLocales
  method, 410
Resources
  annotations for managing, 472
  hierarchy of, 409
  injection, 472
@Resource annotation, 470
Response headers, 262–263
Response page, 268
ResultSet
  interface, 226–227, 349
Result sets (databases)
  accessing columns in, 299
  analyzing, 298
  closing, 302
  for multiple queries, 301
  iterating over rows in, 321
  metadata for, 334
  numbering rows in, 323
  order of rows in, 299
  retrieving multiple, 319–320
  scrollable, 321–324
  updatable, 321, 324–328
ResultSet interface, 329
  absolute method, 323, 327
  beforeFirst, afterLast methods, 323, 327
  cancelRowUpdates method, 325, 328
  close method, 301
  concurrency values, 322, 324, 326, 328
  deleteRow method, 325, 328
  findColumn method, 301
  first, last methods, 323, 327
  getBlob, getClob methods, 316–317
  getMetaData method, 334, 343
  getRow method, 323, 327
  getType method, 323, 326
  getWarnings method, 304
  getXxx methods, 299, 301
  insertRow method, 325, 328
  isClosed method, 301
  isFirst, isLast, isBeforeFirst, isAfterLast
    methods, 324, 327
  iteration protocol, 298
  moveToXxxRow methods, 325, 327
  next method, 298, 300, 321
  previous method, 323, 327
  relative method, 323, 327
  type values, 322, 324, 326, 328
  updateObject method, 301
  updateXxx methods, 324–325, 328
ResultSetMetaData
  interface, 334
  getColumnXxx methods, 334, 343
Retention policies, 472
@Retention annotation, 456, 470, 472
retire/Retire.java, 415
retire/RetireResources_de.java, 427
retire/RetireResources_zh.java, 427
retire/RetireResources.java, 426
retire/RetireStrings_de.properties, 428
retire/RetireStrings_zh.properties, 428
retire/RetireStrings.properties, 428
Retirement calculator example, 413–428
RETURN_GENERATED_KEYS field (Statement), 321
Rewind method (Buffer), 125–126
RFC 2368, 927
RFC 2396, 258
RFC 2616, 261
RFC 2911, 885
RFC 821, 278
RGB color model, 807, 836
Rivest, Ronald, 548
Role-based authentication, 537
rollback method (Connection), 344–346
Root certificate, 563
Root component (file system), 101
Root element (XML), 147, 154
  referencing schemas in, 172
Root node (Swing), 639–647
  handles for, 646–647
  horizontal lines separating children of, 646
rotate method (Graphics2D), 800, 804
Rotation, 800
  and interpolating pixels, 818, 843
  with center point, 801
Round cap, round join, 788
Round join, 789
Round rectangles, 772
RoundEnvironment interface, 477
Round_Rectangle2D class, 769, 772
Round_Rectangle2D.Double class, 785
Row sets (databases), 328–333
  cached, 329–334
  constructing, 329
  modifying, 330
  page size of, 330
RowFilter class, 615–617
  include method, 616, 624
  xxxFilter methods, 615, 625
RowFilter.Entry class, 616
RowId interface, 348
ROWID data type (SQL), 348
Rows (databases), 285
  deleting/inserting, 325
  iterating through, 324
  order of, in result set, 299
  retrieving, 348
  selecting, 288
  updating, 325
Rows (Swing)
  filtering, 615–617
  height of, 612
  hiding, 616
  margins of, 612
  position, in a node, 652
  resizing, 612
  selecting, 602, 612
  sorting, 602, 614–615
  turning off, 614
RowSet interface, 328–332
  execute method, 330, 332
  get/setCommand methods, 330, 332
  get/setPassword methods, 330–331
  get/setURL methods, 330–331
  get/setUsername methods, 330–331
RowSetFactory interface, createXxxRowSet methods, 329, 333
RowSetProvider class, newFactory method, 329, 332
RSA algorithm, 551, 576
rsa/RSATest.java, 577
rt.jar file, 493–494
RTF (Rich Text Format), 712
Ruby programming language, 430
run method
  of PrivilegedAction, 532, 536
  of PrivilegedExceptionAction, 537
  of Tool, 447
Runnable interface, 245
Runtime class, exit method, 509
runtimeAnnotations/ActionlistenerFor.java, 461
runtimeAnnotations/ActionlistenerInstaller.java, 459
RuntimePermission class, 518
S
$ (short), type code, 87, 961
\s, \S, in regular expressions, 131
@SafeVarargs annotation, 470
Sample values, 835
Sandbox, 510–514
SASL (Simple Authentication and Security Layer), 579
Save points (databases), 345
Savepoint interface, getSavepointXxx methods, 347
SAX (Simple API for XML) parser, 149, 199–205
  activating namespace processing in, 201
sax/SAXTest.java, 202
SAXParseException class, getXxxNumber methods, 171
SAXParser class, parse method, 201, 203
SAXParserFactory class
  is/setNamespaceAware methods, 203
  is/setValidating methods, 203
  newInstance, newSAXParser methods, 201, 203
  setFeature method, 202
SAXResult class, 227
SAXSource class, 226–227
  constructor, 231
Scalar functions, 318–319
scale method (Graphics2D), 800, 804
Scaling, 800
  and interpolating pixels, 818, 820
Scanner class, 62, 250
  useLocale method, 380
Scheduling applications
and time zones, 355, 361
computing dates for, 358–360
schema element (XML Schema), 174
Schemas, 342
Script engines, 430–431
adding variable bindings to, 432
implementing Java interfaces, 435
invoking, 430
invoking functions in, 435–436
script/ScriptTest.java, 439
ScriptContext interface, 433
getXxx/setXxx methods of, 434
ScriptEngine interface
createBindings method, 433
eval method, 431–433
get, put methods, 433
getContext method, 434
ScriptEngineFactory interface
getExtensions method, 431
getMethodCallSyntax method, 435
getMimeTypes method, 431
getNames method, 431
ScriptEngineManager class
get method, 433
getEngineXxx methods, 431
put method, 433
Scripting languages, 430–442
advantages of, 430
supported, 430
Scripts
accessing classes through Java interfaces in, 436
compiling, 437
executing, 437
concurrently in multiple threads, 432
invoking, 431
multiple, 432
redirecting I/O of, 434
using Java method call syntax in, 435
Scroll pane (Swing)
with lists, 582, 590
with tabbed panes, 736
with tables, 601
with trees, 653–654
scrollPathToVisible method (JTree), 653, 657
Seconds, leap, 352
Secret key, generating, 570
SecretKey interface, 569
SecretKeySpec class, 574
Secure random generator, 570
SecureRandom class, setSeed method, 570
Securing Java (McGraw/Felten), 513
Security, 491–579
bytecode verification, 504–508
class loaders, 492–508
code signing, 561–567
different levels of, 547
digital signatures, 546–567
encryption, 567–579
user authentication, 530–546
Security managers, 509–530
custom, 510
Security policy, 510
SecurityException, 510, 512
SecurityManager class
checkExit method, 509, 512
checkPermission method, 512–513, 523–524
checkRead method, 513
SecurityPermission class, 519
seek method (RandomAccessFile), 72, 76
“Seek forward only” mode (ImageInputStream), 825
select attribute (XSLT), 225
SELECT statement (SQL), 288–289
executing, 298
for LOBs, 316
multiple, in a query, 319
not supported in batch updates, 345
Selection models, 612
separator constant (File), 56
Separators (file system), 56, 101
sequence element (XML Schema), 173
Serial numbers, 82–83
vs. memory addresses, 84
serialClone/SerialCloneTest.java, 98
Serializable class, 98
Serializable interface, 81, 87, 474
readResolve method, 95
@Serializable annotation, 474
SerializablePermission class, 519
Serialization, 80–100
cloning with, 98–100
file format for, 85–92
MIME type for objects in, 893
modifying default, 92–94
of singletons, 94–95
serial numbers for, 82–83
serialTransfer/SerialTransferFrame.java, 900
serialver program, 96
serialVersionUID constant, 96
server/EchoServer.java, 243
Servers
accessing, 257–277
connecting clients to, 236–238
implementing, 241–249
invoking programs, 268
Server-side programs, 267–277
ServerSocket class, 241–249
accept method, 242, 245
close method, 245
constructor, 245
SERVICEFormatted class (DocFlavor), 875
Service provider interfaces, 824
Servlets, 268, 449–454
hot deployment of, 496
Session class, setDebug method, 279
Set interface, containsAll, equals methods, 525
setItem.java, 485
set/Test.java, 486
setAllowsChildren, setAsksAllowChildren methods
(DefaultMutableTreeNode), 648, 650
setAllowUserInteraction method (URLConnection), 259, 261, 266
setAttribute, setAttributeNS methods (Element), 209, 212
setAutoCommit method (Connection), 346
setAutoCreateRowSorter method (JTable), 602, 604, 614
setAutoSizeResizeMode method (JTable), 611, 622
setBinaryStream method (Blob), 317
SetBooleanArrayRegion function (C), 973
SetBooleanField function (C), 961
setBottomComponent method (JSplitPane), 735
SetByteArrayRegion function (C), 973–974
SetCharArrayRegion function (C), 973–974
SetCharArrayRegion function (C), 973–974
SetCharacterStream method (Clb), 318
SetCharArrayRegion function (C), 973–974
SetCharArrayField function (C), 961
SetClip method (Graphics), 805–806, 854
setClosable method (JInternalFrame), 758
setClosed method (JInternalFrame), 748, 758
setClosedIcon method (DefaultTreeCellRenderer), 664
setColumnSelectionAllowed method (JTable), 613, 622
setCommand method (RowSet), 330, 332
setComparator method (DefaultRowSorter), 614, 624
setComponentAt method (JTabbedPane), 740
setComposite method (Graphics2D), 767, 809, 816
setConnectTimeout method (URLConnection), 259, 266
setContentHandler method (XMLReader), 232
setContentPane method (JInternalFrame), 758
setContents method (Clipboard), 889, 891
setContextClassLoader method (Thread), 495, 503
setContinuousLayout method (JSplitPane), 733, 735
setCrc method (ZipEntry), 79
setCurrency method (NumberFormat), 384
setDateElements method (XmlWriter), 837, 841
setDate method (PreparedStatement), 310, 316
setDebug method (Session), 279
setDecomposition method (Collator), 399
setDefault method (Locale), 376–377
setDefaultNamespace method (XmlStreamWriter), 221
setDefaultRenderer method (JTable), 626
double method (PreparedStatement), 310, 316
setDouble method (Session), 279
setDragEnabled method (of Swing components), 904, 909–910
setDragMode method (JDesktopPane), 751, 757
setDropAction method (TransferSupport), 919
setDropMode method (of Swing components), 913, 919
setEditable method, 714
setEditor method (JSpinner), 710
setEntityResolver method (DocumentBuilder), 164, 170
setErrorHandler method (DocumentBuilder), 170
setErrorWriter method (ScriptContext), 434
setFeature method (SAXParserFactory), 202
setFillForegroundColor method (JTable), 602, 604
setFixedCell methods (JList), 593
setFloatArrayRegion method (JTable), 627, 637
setFloatField function (C), 961
setFloatField function (C), 961
setFocusLostBehavior method (JFormattedTextField), 688, 701
setFrameIcon method (JInternalFrame), 743, 759
setFrom method (MimeMessage), 278
setGroupingUsed method (NumberFormat), 383
setHeaderXxx methods (TableColumn), 627, 637
setIcon method (JInternalFrame), 758
setIconAt method (JTabbedPane), 740
setIconifiable method (JInternalFrame), 758
setIfModifiedSince method (URLConnection), 259, 261, 266
setIgnoringElementContentWhitespace method
(DocumentBuilderFactory), 169, 171
setImage, setImageAutoSize methods (TrayIcon), 936
setImageURL method (SplashScreen), 926
setIndeterminate method (JRootPane), 720, 730
setInput method (ImageReader), 832
setInt method (PreparedStatement), 310, 316
SetIntArrayRegion method (C), 973–974
SetInvalidCharacters method (MaskFormatter), 692, 703
setLayoutOrientation method (List), 587
setLeafIcon method (DefaultTreeCellRenderer), 664
setLeftComponent method (JSplitPane), 735
setLenient method (DateFormat), 78
setLocale method (MessageFormat), 402
setLogWriter method (DriverManager), 296
SetLongArrayRegion method (C), 973–974
SetLongField method (C), 961
setMaxWidth method (DefaultFormatter), 692, 703
setMaxWidth method (TableColumn), 611, 623
setMethod method
of JTable, 79
of JInternalFrame, 719, 730
setMinimumXXXDigits, setMinimumXXXDigits methods
(NumberFormat), 383
setMaximum method
of JInternalFrame, 745, 758
of JProgressbar, 719, 730
setMaxWidth method (TableColumn), 611, 623
setModel method
of JList, 79
of JInternalFrame, 719, 730
setMnemonicAt method (JTabbedPane), 737, 741
setName method (NameCallback), 545
setNameSpaceAware method
of DocumentBuilderFactory, 174, 198–199, 201, 209
of SAXParserFactory, 203
setNote method (ProgressMonitor), 731
SetObjectArrayElement function (C), 972, 974–975
SetObjectField function (C), 956, 961
setOneTouchExpandable method (JSplitPane), 732, 734
setOpenIcon method (DefaultTreeCellRenderer), 664
setOutput method (ImageWriter), 834
setOutputProperty method (Transformer), 213
setOverwriteMode method (DefaultFormatter), 702
setPage method (JEditorPane), 714, 718
setPageable method (PrinterJob), 862, 873
setPageSize method (CachedRowSet), 330, 332
setPaint method (Graphics2D), 767, 797–798
setParseIntegerOnly method (NumberFormat), 383
setPassword method
of PasswordCallback, 546
of RowSet, 330–331
setPixel, setPixels methods (WritableRaster), 835, 841
setPlaceholder method (MaskFormatter), 703
setPlaceholderCharacter method (MaskFormatter), 693, 703
setPopupMenu method (TrayIcon), 936
setPreferredSize method (Component), 685
setPreferredSize method (TableColumn), 611, 623
setPrefix method (XMLStreamWriter), 611, 623
setPrintable method (PrinterJob), 861
setProgress method (ProgressMonitor), 723, 731
setProperty method (XMLInputFactory), 206–207
setPrototypeCellValue method (List), 593
setReader method (ScriptContext), 434
setRenderingHint method (URLConnection), 259, 266
setRenderingHints methods
(Graphics2D), 766, 817–819, 823
setRequestProperty method (URLConnection), 259, 261, 266
setResizable method
of JInternalFrame, 758
of TableColumn, 611, 623
setRightComponent method (JSplitPane), 735
setRootVisible method (JTree), 647, 649
setRootVisible method (DefaultRowSorter), 615, 624
setSelectionHeight, setSelectionMargin, setSelectionAllowed
methods (JTable), 612, 622
setRowSorter method (JTable), 614, 623
Sets
comparing, 525
flattening, 10
setSavepoint method (Connection), 346
setSecurityManager method (System), 516
setSeed method (SecureRandom), 570
setSelected method (JInternalFrame), 744, 758
setSelectedIndex method (JTabbedPane), 736, 740
setSelectionMode method
of List, 583, 587
of ListSelectionModel, 612, 624
SetIntArrayRegion function (C), 973–974
SetShortField method (C), 961
setShowsRootHandles method (JTree), 646, 649
setSize method (ZipEntry), 79
setSortable method (DefaultRowSorter), 614, 624
setSoTimeout method (Socket), 238–239
SetStaticXxxField functions (C), 960–961
setStrength method (Collator), 399
setString method (PreparedStatement), 310, 316
setString, setStringPainted methods (JProgressBar), 720, 730
setStringConverter method (TableRowSorter), 624
setStroke method (Graphics2D), 767, 788, 796
setSubject method (MimeMessage), 278
setTableName method (CachedRowSet), 331–332
setTabXxx methods (JTabbedPane), 737, 741
setText method of JFormattedTextField, 688
of MimeMessage, 278
setTitleAt method (JTabbedPane), 740
setTooltip method (TrayIcon), 936
setTopComponent method (JSplitPane), 735
setToXxx methods (AffineTransform), 802, 804
setTransferHandler method of JComponent, 906, 909
of JFrame, 919
setTransform method (Graphics2D), 802, 804
setURL method (RowSet), 330–331
setUseCaches method (URLConnection), 259, 261, 266
setUsername method (RowSet), 330–331
setScaleXxx method (MutableTreeNode), 642, 649
setValidating method
of DocumentBuilderFactory, 169, 171
of SAXParserFactory, 203
setValidCharacters method (MaskFormatter), 692, 703
setValue method
of AbstractSpinnerModel, 705, 712
of JFormattedTextField, 687–688, 701
of JProgressBar, 719, 730
of JSpinner, 710
of win32RegKey, 988
setValueAsT method (TableModel), 608, 632
setValueContainsLiteralCharacters method (MaskFormatter), 703
setVisible method (JInternalFrame), 744, 759
setVisibleRowCount method (JList), 583, 587
setWidth method (TableColumn), 611, 623
setWriter method (ScriptContext), 434
SGML (Standard Generalized Markup Language), 145
SHA-1 algorithm, 86, 547–549
Shape interface, 769, 787
shape/ShapeTest.java, 777
ShapeMaker class
getPointCount method, 776
makeShape method, 776–777
ShapePanel class, 777
Shapes, 769–786
antialiasing, 817
clipping, 766, 805–807
combining, 786–788
color points of, 776–777
creating, 767
drawing, 766–769
filling, 766–767, 797
rendering, 768
transformations of, 766
Shared libraries, 946, 985
initialization blocks in, 947
short type
printing, 61
type code for, 87, 961
vs. C types, 947
writing in binary format, 69
ShortBuffer class, 124
ShortLookupTable class, 844, 851
shouldSelectCell method (CellEditor), 631, 638
show method (JInternalFrame), 759
showInternalXxxDialog methods (JOptionPane), 750
shutdownXxx methods (Socket), 249
Side files, 475
Signatures, 546–567
generating, 962
mangling, 961–963
signed/FileReadApplet.java, 566
Simple types, 172
SimpleDateFormat class, toPattern method, 712
SimpleDoc class, 874, 878
SimpleFileVisitor class, 113
visitFile, visitFileFailed methods, 113, 115
xxvVisitDirectory methods, 115
SimpleJavaFileObject class
extending, 445
generateContent method, 445, 449
openOutputStream method, 445, 449
SimpleLoginModule class, checkLogin, initialize methods, 538
SimplePrincipal class, 433
simpleType element (XML Schema), 172
SimulatedActivity class, 720
Single value annotations, 465
Singletons
of the splash screen, 923
serializing, 94–95
SISC Scheme programming language, 430, 438–439
size method
of BasicFileAttributes, 109
of Files, 108–109
skip method
of InputStream, 50
of Stream, 10–11
skipBytes method (DataInput), 71
SMALLINT data type (SQL), 291, 348–349
SMTP (Simple Mail Transport Protocol), 277–280
Socket class
connect method, 239
constructor, 238–239
getInputStream method, 237–238, 242
getOutputStream method, 238, 242
isClosed, isConnected methods, 239
isXxxxShutdown, shutdownXxxx methods, 249
setSoTimeout method, 238–239
Socket permissions, 521
socket/SocketTest.java, 236
SocketChannel class, 250
open method, 250, 256
SocketPermission class, 517
Sockets
half-closing, 249
interrupting, 250–257
opening, 237, 509
timeouts, 238–239
SocketTimeoutException, 238, 266
Software updates, 932
Solaris operating system
compiling invocation API, 984
using Sun compiler, 943
sort method (Collections), 394
sorted method (Stream), 11–12
Source interface, 226, 349
Source files
character encoding of, 407–408
locating, 444
reading from disk, 444
Source-level annotations, 475–480
Space. See Whitespace
SPARC processor, big-endian order in, 70
spinner/PermutationSpinnerModel.java, 708
spinner/SpinnerFrame.java, 706
SpinnerDateModel class, 711
SpinnerListModel class, 704, 711
SpinnerNumberModel class, 704, 711
Spinners (Swing), 703–712
increment value in, 704
traversal order in, 704–705
Splash screens, 921–926
drawing on, 922–923
replacing with follow-up window, 923
SplashScreen class
close method, 926
createGraphics method, 926
get/setImageURL methods, 926
getBounds method, 926
getSplashScreen method, 923, 926
update method, 922, 926
splashScreen/SplashScreenTest.java, 923
split method
of Pattern, 139–140
of String, 64
Split panes (Swing), 732–735
splitAsStream method (Pattern), 6, 8
splitPane/SplitPaneFrame.java, 733
sprintf function (C), 954
split function (C), 953
SQL (Structured Query Language), 285–290
changing data inside databases, 290
commands in, 293
data types in, 291, 347–349
equality testing in, 289
escapes in, 318–319
exceptions in, 302–305
executing statements in, 298–301
keywords in, 288
reading instructions from a file, 305
strings in, 289
vs. Java, 311
warnings in, 303
wildcards in, 289
SQLException class, 302–305, 323
and rolling back, 344
and save points, 347
getXxxx, iterator methods, 302, 304
SQLPermission class, 520
SQLWarning class, 303, 323
getNextWarning method, 304
SQLXML data type (SQL), 348–349
Square cap, 788–789
Square root, computing, 17
SQuirreL program, 334
Index

SRC, SRC_Xxx composition rules, 809–810
src.jar file, 981
sRGB standard, 837
SSL (Secure Sockets Layer), 579
Standard extensions, 493
StandardCharsets class, 69
StandardJavaFileManager interface, 444–445
getJavaFileObjectsFromXxx methods, 448
start method (Matcher), 135, 137, 140–141
startDocument method (ContentHandler), 204
startElement method (ContentHandler), 200–204
Statement interface, 298–301
addBatch method, 345–347
close, closeInCompletion methods, 300, 302
execute method, 300, 306, 319, 321
eexecuteBatch method, 346–347
eexecuteLargeBatch method, 347
eexecuteQuery method, 298, 300, 321, 344
getMoreResults method, 320
getResultSet method, 300
getWarnings method, 304
isClosed method, 300
RETURN_GENERATED_KEYS field, 321
using for multiple queries, 301
Statements (databases)
closing, 302
complex, 311
concurrently open, 301
executing, 298–301
 grouping into transactions, 344–347
in batch updates, 345
multiple, 301
prepared, 309–316
truncations in, 303
Static fields, in native code, 960–961
Static initialization blocks, 944
Static methods, calling from native code, 964–965
STAX parser, 205–208, 214–222
namespace processing in, 206
no indented output in, 214
stax/StAXTest.java, 206
StAXSource class, 226
stopCellEditing method (CellEditor), 631–632, 638
Stored procedures, 318–319
stream method
 of Arrays, 8, 36
 of Collection, 2–5
Stream interface
collect method, 19, 22, 35
concat method, 11
count method, 3–4, 12
distinct method, 11–12, 43
empty method, 5, 8
filter method, 3–4, 9–10, 13
findAny method, 13
findXxx methods, 13
flatMap method, 10
forEach method, 19, 22
generate method, 5, 8, 36
iterate method, 5, 8, 12, 36
limit method, 10–11, 43
map method, 9–10
max, min methods, 12
of method, 5, 8
peek method, 11–12
reduce method, 33–35
skip method, 10–11
sorted method, 11–12
toArray method, 19, 22
xxxMatch methods, 13
Streaming parsers, 149, 199–208
StreamPrintService class, 878
StreamPrintServiceFactory class, 878
getPrintService method, 878–879
lookupStreamPrintServicefactories method, 879
StreamResult class, 213, 227
Streams (Java SE 8), 1–46
 combining, 11
converting:	 between objects and primitive types of, 36
to arrays, 19
creating, 5–8
debugging, 11
empty, 5, 34
encrypted, 574–575
extracting substreams from, 10
filters for, 726
flattening, 10
for print services, 878–879
infinite, 3, 5, 10, 12
input, 150, 726–731
intermediate and terminal operations for, 3, 12
of primitive type values, 36–41
of random numbers, 37
operations on:
  noninterference of, 43
  threadsafe, 42
  ordered, 42
  parallel, 2, 13, 19, 41–46
  pipeline of, 11
  processed lazily, 3, 12, 43
  reductions of, 12
  sorting, 11
  transformations of, 9–12
  vs. collections, 3

streams/CountLongWords.java, 4
streams/CreatingStreams.java, 6
streams/PrimitiveTypeStreams.java, 37
StreamSource class, 226
  constructor, 231
  transform method, 227
String class, 54
  charAt method, 9
  compareTo method, 393
  split method, 64
  toLowerCase method, 9
  trim method, 153, 379
STRING class (DocFlavor), 875
String parameters, 949–956
StringBuilder class, 54, 124
StringBuilder class, 54, 73, 688, 727
Strings
  encoding, 407
    in Unicode, 371
    fixed-size, I/O of, 73–74
    in native code, 949–956
    in SQL, 289
    internationalizing, 408–410
    ordering, 393
    patterns for, 128–141
    permutations of, 705–706
    printing, 61
    sorting, 394
    splitting, 6
    transforming to lowercase, 9
    writing in binary format, 69
StringSelection class, 888, 894
stringToValue method (AbstractFormatter), 693, 701
Stroke interface, 788
stroke/StrokeTest.java, 791
StrokePanel class, 791
Strokes, 766, 788–796
  antialiasing, 818
  dash patterns of, 790
end cap styles of, 788–790
join styles of, 788–789
setting, 767
thickness of, 788
StyledDocument interface, 681
StylePad demo, 682
Stylesheets (XSLT), 222–232
Subject class
  doAs, doAsPrivileged methods, 532–533, 536
  getPrincipals method, 536
Subjects (logins), 532
subSequence method (CharSequence), 55
subtract method (Area), 787–788
Subtrees (Swing), 644, 661
  adding nodes to, 653
  collapsed and expanded, 645
Suetonius, Gaius Tranquillus, 498
sufficient keyword, 532
sun method (XxxStream), 37, 39–40
summarizingXxx methods (Collectors), 20, 23
summaryStatistics method (XxxSummaryStatistics), 37, 39–40
summingXxx methods (Collectors), 29, 33
Sun compiler, 943
SunJCE ciphers, 568
Superclasses, type use annotations in, 468
Supplier interface, get method, 8
@SupportedAnnotationTypes annotation, 476
SupportedValuesAttribute interface, 879
supportsBatchUpdates method (DatabaseMetaData), 345, 347
supportsResultSet Xxx methods (DatabaseMetaData), 323, 328
@SuppressWarnings annotation, 470–471
SVG (Scalable Vector Graphics), 211–212
Swing, 581–764
data transfer in, 904–921
generating dynamic code for, 449
loading time in, 921
Swing components
  attaching verifiers to, 690
desktop panes, 741–760
drag-and-drop behavior of, 909–912
ditor pane, 712–719
internal frames, 741–760
layers, 760–764
layout managers for, 175–189
lists, 582–588
Swing components (continued)
organizing, 731–764
progress bars, 719–722
progress monitors, 722–726
spinners, 703–712
split panes, 732–735
tabbed panes, 735–741
tables, 599–638
text, 681–719
trees, 639–680
Swing Connection newsletter, 694
SwingWorker class, 720
Symmetric ciphers, 567–569
performance of, 576
SyncProviderException interface, 331–332
System class
console method, 406
loadLibrary method, 944, 946
setSecurityManager method, 516
SYSTEM identifier (DTD), 164, 210
System class loader, 493
System properties, in policy files, 521
System tray, 932–937
System.err class, 61
System.in class, 61
and character encoding, 406
System.out class, 61
and character encoding, 406
println method, 405–406
SystemTray class, 932–937
add method, 932, 936
getSystemTray method, 932, 936
getTrayIconSize method, 936
isSupported method, 932, 936
remove method, 936
systemTray/SystemTrayTest.java, 934
Table
t literal (SQL), 318
\t, in regular expressions, 130
Tabbed panes (Swing), 735–741
adding/removing tabs in, 736
labels of, 737
scrolling, 736
tabbedPane/TabbedPaneFrame.java, 738
Table cell renderers, 609, 626
Table index values, 614
Table models (Swing), 600, 604–608
updating after cells were edited, 632
table/TableTest.java, 603
TableCellEditor interface, 629, 631
getTableCellEditorComponent method, 629, 631, 638
tableCellRender/ColorTableCellEditor.java, 635
tableCellRender/ColorTableCellRenderer.java, 635
tableCellRender/PlanetTableModel.java, 633
tableCellRender/TableCellRendererFrame.java, 632
TableCellRenderer interface
getTableCellRendererComponent method, 626, 637
implementing, 626
TableCellModel interface, 610
getColumn method, 623
TableModel interface, 614
getColumnAt methods, 605–606, 608, 632
columnClass method, 609, 621
columnName method, 606, 608
getCellCount methods, 605–606, 608
implementing, 605
isCellEditable method, 608, 628
tableModel/InvestmentTable.java, 606
tableRowColumn/PlanetTableFrame.java, 617
TableRowSorter class, 614
setStringConverter method, 624
Tables (databases), 285
changing data in, 290
creating, 290
duplication of data in, 287
inspecting, 287
metadata for, 333
multiple, selecting data from, 289
removing, 295
Tables (Swing), 599–638
cells in:
editing, 628–629
rendering, 626
selecting, 612–614, 626
columns in:
accessing, 610
adding, 617
hiding, 617–625
naming, 606
rearranging, 601
rendering, 609–610
resizing, 602, 611–612
selecting, 612
constructing, 601, 605
headers in, 601
derendering, 627
printing, 602
relationship between classes of, 610
rows in:
filtering, 615–617
hiding, 616
margins of, 612
resizing, 612
selecting, 602, 612
sorting, 602, 614–615
scrolling, 601
TableStringConverter class, toString method, 615, 624
@Target annotation, 456, 470, 472–473
TCP (Transmission Control Protocol), 238
telnet, 233–236
activating/connecting, 234
surface windows communicating simultaneously, 246–247
template element (XSLT), 224
Temporal interface, with method, 359
TemporalAdjuster interface, 359
TemporalAdjusters class, 358–359
test/Test08.java, 296
@Test annotation, 455
@TestCase, @TestCases annotations, 475
Text, 60–69
copying, 887
drag and drop of, 904–905
encoding of, 67–69
formatting, 887
generating from XML files, 225–227
output, 60–62
printing, 852–862, 874
reading, 62–63
saving objects in, 63–67
transferring via clipboard, 888–892
transmitting through sockets, 241–249
vs. binary data, 60
Text components (Swing), 681–719
hierarchy of, 681
monitoring input in, 682–685
styled, 681, 712
Text fields
committing, 687
editing, 629
formatted, 685–703
input in:
integers, 686
turning to upper/lower case, 690
validating, 524–530, 683, 685–703
losing focus, 687–688
reverting, 687
Text files, encoding of, 405–406
Text nodes (XML)
constructing, 209
retrieving from XML, 153
TextCallbackHandler class, 538
textChange/ColorFrame.java, 683
textFile/TextFileTest.java, 65
textFormat/FormatTestFrame.java, 695
textFormat/IntFilter.java, 698
textFormat/IPAddressFormatter.java, 699
textLayout class, getXXX methods, 807
TextStyle class, 882
TexturePaint class, 797–799
this keyword, 956
annotating, 468–469
Thread class, get/setContextClassLoader methods, 495, 503
threaded/ThreadEchoServer.java, 247
ThreadedEchoHandler class, 245–248
Threads
blocking, 49, 250–257, 723
executing scripts in, 432
for new pages in JEditorPane, 715
Internet connections with, 245–248
race conditions in, 42
referencing class loaders in, 495–496
Three-tier model, 284–285
Throw, ThrowNew functions (C), 974–975, 979
Throwable class, 974
Thumbnails, 826
Tiling windows, 744, 746–747
Time
current, 352
editing, 691
formatting, 365–368, 385–393
literals for, 318
local, 360–361
measuring, 352–353
parsing, 367
zoned, 361–365
Time class, 369
Time of day service, 234
Time pickers, 705
Time zones, 385
TIME, TIMESTAMP data types (SQL), 291, 318, 348
timeline/timeline.java, 354
Timeouts, 238–239
Timer class, 723
Timestamp class, 369
Timestamps, 365
using instants as, 352
TimeZone class, and legacy classes, 369
toAbsolutePath method (Path), 102–103
toArray method
  of AttributeSet, 886
  of streams, 19, 22, 36, 38–39–40
toCollection method (Collectors), 20, 23
toConcurrentMap method (Collectors), 25, 27
toDays method (Duration), 353
toFile method (Path), 102–103
toFormat method (DateTimeFormatter), 367
toHours method (Duration), 353
toInstant method of Date, 369
  of ZonedDateTime, 361, 363
toLanguageTag method (Locale), 375, 377
toList method (Collectors), 20, 23
toLocalXxx methods (ZonedDateTime), 363
toLowerCase method (String), 9
toMap method (Collectors), 24–27
toMillis, toMinutes, toNanos methods (Duration), 353
toNanosOfDay, toSecondsOfDay methods (LocalTime), 360
tool interface, run method, 447
Toolkit class, getSystemClipboard method, 888, 891
ToolProvider class, getSystemJavaCompiler method, 443
tools.jar file, 443
Tooltips, for tray icons, 932
toPath method (File), 102–103
toPattern method (SimpleDateFormat), 712
Top-level windows, opening, 509
toSeconds method (Duration), 353
toSet method (Collectors), 20, 23, 29
toString method
  implementing with annotations, 477–480
  of Annotation, 464
  of CharSequence, 55
  of Currency, 385
  of DefaultFormatter, 691
  of Locale, 377
  of TableStringConverter, 615, 624
  of Variable, 674
toZonedDateTime method (GregorianCalendar), 369
Transactions, 344–347
  committing/rolling back, 344
  error handling in, 346
Transfer handlers, 906–907
  adding, 910
Transfer wrapper, 899
Transferable interface, 892–894
  getTransferData method, 892
  isDataFlavorSupported method, 892
TransferHandler class, 909–912
  canImport method, 912, 918
  constructor, 909
  createTransferable method, 909, 912
  exportAsDrag method, 910, 912
  exportDone method, 911–912
  getTransferData method, 909, 912
  importData method, 912–914, 918
TransferHandler.DropLocation class, 914
  getDropPoint method, 920
TransferSupport class
  get/setDropAction methods, 919
  getComponent, getDataFlavors methods, 919
  getDropLocation method, 914, 919
  getTransferData method, 919
  getTransferable method, 913
  getUserDropAction method, 919
  isDrop method, 914, 919
transferText/TextTransferFrame.java, 890
transform method
  of Graphics2D, 767, 802, 804
  of StreamSource, 227
  of Transformer, 213, 226
  transform/makehtml.xsl, 227
  transform/makeprop.xsl, 228
  transform/TransformTest.java, 228
Transformations, 766, 799–804
  affine, 802, 843
  composing, 800–801
  fundamental types of, 800
  matrices for, 802
  order of, 800
  setting, 767
  using for printing, 864
Transformer class
  setOutputProperty method, 213
  transform method, 213, 226
TransformerFactory class
- newInstance method, 213
- newTransformer method, 213, 231
- transient, 92
- Transitional events, 584
- translate, 800, 804, 864
- Translation, 800
- Transparency, 807–817
- Traversal order
  - in a spinner, 704–705
  - in a tree:
    - breadth-first vs. depth-first, 659
    - postorder, 659
- Tray icons, 932–933
  - pop-up menus for, 933
  - tooltips for, 932
- TrayIcon class
  - add/removeActionListener methods, 937
  - constructor, 936
  - displayMessage method, 937
  - get/setImage methods, 936
  - get/setPopupMenu methods, 936
  - get/setTooltip methods, 936
  - is/setImageAutoSize methods, 937
- Tree events, 664–671
- Tree models
  - constructing, 641, 672
  - custom, 671–680
  - default, 641
- Tree parsers, 149
- Tree paths, 650–658
  - constructing, 653, 660
- Tree selection listeners, 664
  - adding to a tree, 664
- TreeSelectionEvent class
  - getPath method, 671
  - getPaths method, 665, 671
- TreeSelectionListener interface
  - implementing, 664–671
  - valueChanged method, 664, 666, 671
- TreeSelectionModel interface
  - getPath method, 671
  - getPaths method, 665, 671
- TreeSelectionListener interface
  - implementing, 664–671
  - valueChanged method, 664, 666, 671
- TreeSelectionModel interface, 665
- trim method (String), 153, 379
- Troubleshooting, See Debugging
- True Odds: How Risks Affect Your Everyday Life (Walsh), 547
tryLock method (FileChannel) method (Lock), 126–128
try-with-resources statement, 53, 112
  closing files with, 110
  for database connections, 302
  with locks, 127
ts literal (SQL), 318
Type bounds, type use annotations in, 468
TYPE_BYTE_GRAY field (BufferedImage), 838
Type codes, 87, 961
Type definitions, 172
  anonymous, 173
  nesting, 173
TYPE_INT_ARGB field (BufferedImage), 835–836, 840
Type parameters, annotating, 466
Type use annotations, 467
TYPE_XXX fields (AffineTransformOp), 843, 850
TypeElement interface, 477
Types. See Data types
Typesafe enumerations, 94–95

U
  \u in masks, 692
  \w, in regular expressions, 130
UDP (User Datagram Protocol), 238
UIManager class, 626
Unicode standard
  and input/output streams, 48
  and native code, 950
  character order in, 393
  converting to binary data, 60
  in property files, 410
  in regular expressions, 133
  normalization forms in, 394
  using for all strings, 371
uninstallUI method (LayerUI), 764
Units of measurement, 148
UNIX operating system
  authentications in, 531
  file names starting with a period in, 516
  Java deployment directory in, 564
  line feed in, 61, 405
  paths in, 101
UnixLoginModule class, 532
UnixNumericGroupPrincipal class, 532
UnixPrincipal class, 531–532
  getName method, 531
UnknownHostException, 237
unordered method (BaseStream), 43, 45
Unparsed external entities, 168
unread method (PushbackInputStream), 59
UnsatisfiedLinkError, 941
UnsupportedFlavorException, 892
until method (LocalDate), 356–357
update method
  of Cipher, 568, 571, 573, 575
  of MessageDigest, 549–550
  of SplashScreen, 922, 926
UPDATE statement (SQL), 290, 310, 324
  executing, 298, 300, 316
  in batch updates, 345
  truncations in, 303
  vs. methods of ResultSet, 325
updateObject method (ResultSet), 301
updateXxx methods (ResultSet), 324–325, 328
Upper case, turning characters to, 690
URI class
  getXxx methods, 258
  nomailto support in, 927
  no resource accessing with, 257
URLs (Uniform Resource Identifiers), 257
  absolute vs. relative, 258
  base, 259
  for e-mails, 927
  hierarchical, 258
  namespace, 196–199
  opaque vs. nonopaque, 258
  schemes for, 258
URISyntax class, 882
URL class (DocFlavor), 875
URL class (java.lang.Object), 257–259
  accepted schemes for, 257
  openConnection method, 259, 265
  openStream method, 257, 265
URLClassLoader class, 493
  constructor, 503
  loadClass method, 494
URLConnection class, 257, 259–267
  connect method, 259, 262, 266
  getSetAllowUserInteraction methods, 259, 261, 266
  getSetConnectTimeout methods, 259, 266
  getSetDoInput, getSetDoOutput methods, 259–260, 265
  getSetModifiedSince methods, 259, 261, 266
  getSetReadTimeout methods, 259, 266
  getSetRequestProperty methods, 259, 261, 266
  getSetUseCaches methods, 259, 261, 266
  getContent method, 267
value-of element (XSLT), 225
Values  
  grouping, 28–30  
  partitioning, 28
valueToString method (AbstractFormatter), 693, 701
VARCHAR data type (SQL), 291, 348
Variable class, 673
  toString method, 674
VariableElement interface, 477
Variables  
  annotating, 456, 467  
  binding, 432  
  fields of, 674  
  initializing, 504  
  scope of, 433
Variants, in locales, 373, 410
Vendor name, of a reader, 825
verifier/VerifierTest.java, 507
Verifiers, 504–508, 690
verify method (InputVerifier), 690
VeriSign, Inc., 555, 557, 561
Version number, of a reader, 825
Versioning, 95–98
VERTICAL_SPLIT value (JSplitPane), 732
VERTICAL, VERTICAL_WRAP values (JList), 583
Vetoable change listeners, 748
VetoableChangeListener interface  
  implementing, 749
  vetoableChange method, 749, 759
Vetoing, 748–750
view/ViewDB.java, 335
visitFile, visitFileFailed methods  
  of FileVisitor, 112  
  of SimpleFileVisitor, 113, 115
Visual representation, 284

W
\w, \W, in regular expressions, 131
walk method (Files), 110
walkFileTree method (Files), 112–114
warning method (ErrorHandler), 170–171
Warnings  
  in applets, 526  
  in SQL, 303  
  suppressing, 471
WBMP format, 823
WeakReference object, 675
Web applications, connection management  
  in, 349–350
Web containers, 472
Web crawlers, 200
  with SAX parser, 201–202
  with StAX parser, 206
Web pages
  dynamic, 449–454
  separating applet class loaders for, 496
WebRowSet interface, 329
WHERE statement (SQL), 289
Whitespace
  ignoring, while parsing, 153
  in e-mail URIs, 270, 927
  in regular expressions, 131, 133
  in XML Schema, 174
Wildcards, type use annotations in, 468
Wildes, Oscar, 372
win32reg/Win32RegKey.c, 993
win32reg/Win32RegKey.java, 991
win32reg/Win32RegKeyTest.java, 1000
Win32RegKey class, 987, 990
  get/setValue methods, 987–988
  names method, 987
Win32RegKeyNameEnumeration class, 989–990
Windows operating system
  activating telnet in, 234
  authentication in, 533
  character encodings in, 404
  classpath in, 292
  compiling invocation API, 984
  cut and paste in, 887
  drag and drop in, 904
  dynamic linking in, 981
  frame dragging in, 751
  glob syntax in, 112
  Java deployment directory in, 564, 566
  line feed in, 61, 405
  multiple document interface of, 741–742
  paths in, 56, 101
  permissions in, 520
  registry, accessing from native code, 985–1001
  resources in, 408
  system tray in, 932
  using Microsoft compiler, 944
Windows look-and-feel
  cascading/tiling in, 744
  trees in, 645
  with method (Temporal), 359
  withLocale method (DateTimeFormatter), 365, 392
  withXxx methods (Date and Time API), 356, 360, 363
WordCheckPermission class, 524–530
Words
  counting, in a book, 2
  in regular expressions, 131
Working directory, 56
wrap method (ByteOutputStream), 123, 125
WritableByteChannel interface, 250
WritableRaster class, 835
  setDataElements method, 837, 841
  setPixel, setPixels methods, 835, 841
write method
  of CipherOutputStream, 575
  of files, 104
  of ImageIO, 824, 831
  of ImageWriter, 827, 834
  of OutputStream, 48, 50–51
write/RectangleComponent.java, 218
write/XMLWriteFrame.java, 215
write/XMLWriteTest.java, 215
writeExternal method (Externalizable), 93–94
writeFixedString method (DataIO), 73–74
writeInsert method (ImageWriter), 827, 834
writeObject method
  of Date, 93
  of ObjectOutputStream, 80, 85, 92
Writer class, 48, 52–53
  write method, 51
writeUTF method (DataOutput), 70, 72
writeXxx methods (DataOutput), 69, 72, 81
writeXxx methods (XMLStreamWriter), 214, 221–222

X
X Window System
  cut and paste in, 887
  frame dragging in, 751
X.509 format, 554
\x, in regular expressions, 130
XHTML (Extensible Hypertext Markup Language), 200
XML (Extensible Markup Language), 143–232
  annotated version of the standard, 146
  case sensitivity of, 146
  end and empty tags in, 146
  hierarchical structures in, 144–145
  in databases, 349
  Java support of, 494
  namespaces in, 196–199
  using for layout description, 175–189
  vs. HTML, 146
XML documents
- DTDs in, 147, 163–164
- format of, 145
- generating, 208–222
  - from non-XML legacy data, 227
- HTML files from, 222–225
- plain text from, 225–227
- locating information in, 190–196
- parsing, 149–162
- structure of, 146–149, 150, 163
- validating, 162–189
  - with/without namespaces, 209–210
- writing, 210–211
  - with StAX, 214–222
XML Schema, 163, 172–174
- attributes in, 174
- parsing with, 174
- referencing in XML documents, 172
- repeated elements in, 174
- type definitions in, 172
  - anonymous, 173
  - nesting, 173
- whitespace in, 174
XMLInputFactory class
- createXMLStreamReader method, 207
- newInstance method, 207, 197
- setProperty method, 206–207
- xmlns attribute (XSLT), 197
XMLOutputFactory class
- createXMLStreamWriter method, 214, 221
- newInstance method, 214, 221
XPath interface, evaluate method, 191, 196
- xpath/XPathTester.java, 192
XPathFactory class
- newInstance method, 191, 195
- newPath method, 195
xs:, xsd: prefixes (XSL Schema), 172
xsd:attribute element (XML Schema), 174
xsd:choice element (XML Schema), 173
xsd:complexType element (XML Schema), 173
xsd:element element (XML Schema), 172
xsd:enumeration element (XML Schema), 172
xsd:schema element (XML Schema), 172
xsd:sequence element (XML Schema), 173
xsd:simpleType element (XML Schema), 172
xsl:apply-templates element (XSLT), 224
xsl:output element (XSLT), 224
xsl:template element (XSLT), 224
xsl:value-of element (XSLT), 225
XSLT (Extensible Stylesheet Language Transformations), 210, 222–232
  - copying attribute values in, 225
  - templates in, 224
XSLT processor, 222
Y
- Year, YearMonth classes, 357
- Years, leap, 356
Z
- z (boolean), type code, 87, 961
- \Z, \z, in regular expressions, 132
- ZIP archives, 77–80
  - reading, 77
  - numbers from, 58
  - writing, 77
- ZipEntry class, methods of, 79
- ZipFile class, methods of, 80
- ZipInputStream class, 51, 77
  - closeEntry, getNextEntry methods, 77–78
  - constructor, 78
  - read method, 77
- ZipOutputStream class, 51, 77
  - closeEntry, putNextEntry methods, 77–78
  - constructor, 78
  - setLevel, setMethod methods, 78
ZonedDateTime class
- and legacy classes, 369
  - methods of, 361–365, 369, 386, 393
  - zonedtimes/ZonedDateTime.java, 364