This page intentionally left blank
<table>
<thead>
<tr>
<th>Chapter 2: XML</th>
<th>93</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Introducing XML</td>
<td>94</td>
</tr>
<tr>
<td>2.1.1 The Structure of an XML Document</td>
<td>96</td>
</tr>
<tr>
<td>2.2 Parsing an XML Document</td>
<td>99</td>
</tr>
<tr>
<td>2.3 Validating XML Documents</td>
<td>113</td>
</tr>
<tr>
<td>2.3.1 Document Type Definitions</td>
<td>114</td>
</tr>
<tr>
<td>2.3.2 XML Schema</td>
<td>122</td>
</tr>
<tr>
<td>2.3.3 A Practical Example</td>
<td>125</td>
</tr>
<tr>
<td>2.4 Locating Information with XPath</td>
<td>140</td>
</tr>
<tr>
<td>2.5 Using Namespaces</td>
<td>147</td>
</tr>
<tr>
<td>2.6 Streaming Parsers</td>
<td>150</td>
</tr>
<tr>
<td>2.6.1 Using the SAX Parser</td>
<td>150</td>
</tr>
<tr>
<td>2.6.2 Using the StAX Parser</td>
<td>156</td>
</tr>
<tr>
<td>2.7 Generating XML Documents</td>
<td>159</td>
</tr>
<tr>
<td>2.7.1 Documents without Namespaces</td>
<td>159</td>
</tr>
<tr>
<td>2.7.2 Documents with Namespaces</td>
<td>160</td>
</tr>
<tr>
<td>2.7.3 Writing Documents</td>
<td>161</td>
</tr>
<tr>
<td>2.7.4 An Example: Generating an SVG File</td>
<td>161</td>
</tr>
<tr>
<td>2.7.5 Writing an XML Document with StAX</td>
<td>164</td>
</tr>
<tr>
<td>2.8 XSL Transformations</td>
<td>173</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 3: Networking</th>
<th>185</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Connecting to a Server</td>
<td>185</td>
</tr>
<tr>
<td>3.1.1 Socket Timeouts</td>
<td>190</td>
</tr>
<tr>
<td>3.1.2 Internet Addresses</td>
<td>192</td>
</tr>
<tr>
<td>3.2 Implementing Servers</td>
<td>194</td>
</tr>
<tr>
<td>3.2.1 Serving Multiple Clients</td>
<td>197</td>
</tr>
<tr>
<td>3.2.2 Half-Close</td>
<td>201</td>
</tr>
<tr>
<td>3.3 Interruptible Sockets</td>
<td>202</td>
</tr>
<tr>
<td>3.4 Getting Web Data</td>
<td>210</td>
</tr>
<tr>
<td>3.4.1 URLs and URIs</td>
<td>210</td>
</tr>
</tbody>
</table>
3.4.2 Using a URLConnection to Retrieve Information .................. 212
3.4.3 Posting Form Data ................................................................. 222
3.5 Sending E-Mail ........................................................................ 230

Chapter 4: Database Programming ............................................ 235

4.1 The Design of JDBC ............................................................... 236
  4.1.1 JDBC Driver Types ............................................................. 236
  4.1.2 Typical Uses of JDBC ......................................................... 238
4.2 The Structured Query Language ............................................. 239
4.3 JDBC Configuration ............................................................... 245
  4.3.1 Database URLs ................................................................. 246
  4.3.2 Driver JAR Files ............................................................... 246
  4.3.3 Starting the Database ........................................................ 247
  4.3.4 Registering the Driver Class .............................................. 248
  4.3.5 Connecting to the Database .............................................. 249
4.4 Executing SQL Statements .................................................... 252
  4.4.1 Managing Connections, Statements, and Result Sets ............ 255
  4.4.2 Analyzing SQL Exceptions ................................................. 256
  4.4.3 Populating a Database ........................................................ 258
4.5 Query Execution ...................................................................... 262
  4.5.1 Prepared Statements .......................................................... 263
  4.5.2 Reading and Writing LOBs ............................................... 269
  4.5.3 SQL Escapes ..................................................................... 271
  4.5.4 Multiple Results ............................................................... 272
  4.5.5 Retrieving Autogenerated Keys ......................................... 273
4.6 Scrollable and Updatable Result Sets ....................................... 274
  4.6.1 Scrollable Result Sets ........................................................ 274
  4.6.2 Updatable Result Sets ........................................................ 277
4.7 Row Sets .................................................................................. 281
  4.7.1 Constructing Row Sets ....................................................... 282
  4.7.2 Cached Row Sets .............................................................. 282
4.8 Metadata .................................................................................. 286
4.9 Transactions ............................................................................ 296
  4.9.1 Save Points ..................................................................... 297
  4.9.2 Batch Updates ................................................................. 298
Chapter 5: Internationalization ................................................................. 305
  5.1 Locales ....................................................................................................... 306
  5.2 Number Formats .......................................................................................... 311
    5.2.1 Currencies .......................................................................................... 318
  5.3 Date and Time ................................................................................................ 319
  5.4 Collation .................................................................................................... 328
    5.4.1 Collation Strength ............................................................................... 329
    5.4.2 Decomposition ................................................................................... 329
  5.5 Message Formatting .................................................................................... 336
    5.5.1 Choice Formats .................................................................................. 338
  5.6 Text Files and Character Sets ..................................................................... 340
    5.6.1 Character Encoding of Source Files ............................................... 340
  5.7 Resource Bundles ...................................................................................... 341
    5.7.1 Locating Resource Bundles ............................................................... 342
    5.7.2 Property Files .................................................................................... 343
    5.7.3 Bundle Classes .................................................................................. 344
  5.8 A Complete Example ............................................................................... 346

Chapter 6: Advanced Swing ......................................................................... 363
  6.1 Lists .......................................................................................................... 364
    6.1.1 The JList Component ........................................................................ 364
    6.1.2 List Models ......................................................................................... 370
    6.1.3 Inserting and Removing Values ......................................................... 375
    6.1.4 Rendering Values ............................................................................... 377
  6.2 Tables .......................................................................................................... 381
    6.2.1 A Simple Table .................................................................................. 382
    6.2.2 Table Models ...................................................................................... 386
    6.2.3 Working with Rows and Columns ..................................................... 390
      6.2.3.1 Column Classes ........................................................................... 390
      6.2.3.2 Accessing Table Columns ........................................................... 392
      6.2.3.3 Resizing Columns ....................................................................... 392
      6.2.3.4 Resizing Rows ............................................................................ 393
      6.2.3.5 Selecting Rows, Columns, and Cells ......................................... 394
6.2.3.6 Sorting Rows ............................................................... 395
6.2.3.7 Filtering Rows ............................................................. 396
6.2.3.8 Hiding and Displaying Columns ............................. 398
6.2.4 Cell Rendering and Editing ...................................................... 408
6.2.4.1 Rendering the Header ................................................ 409
6.2.4.2 Cell Editing .................................................................. 410
6.2.4.3 Custom Editors ........................................................... 411
6.3 Trees .......................................................................................................... 420
6.3.1 Simple Trees ............................................................................... 421
6.3.1.1 Editing Trees and Tree Paths .................................... 431
6.3.2 Node Enumeration ................................................................ 440
6.3.3 Rendering Nodes ....................................................................... 442
6.3.4 Listening to Tree Events ........................................................... 445
6.3.5 Custom Tree Models ....................................................................... 453
6.4 Text Components .................................................................................... 462
6.4.1 Change Tracking in Text Components ................................... 463
6.4.2 Formatted Input Fields ............................................................. 467
6.4.2.1 Integer Input ............................................................... 468
6.4.2.2 Behavior on Loss of Focus ......................................... 468
6.4.2.3 Filters ............................................................................ 470
6.4.2.4 Verifiers ........................................................................ 471
6.4.2.5 Other Standard Formatters ....................................... 472
6.4.2.6 Custom Formatters ..................................................... 474
6.4.3 The JSpinner Component ........................................................... 485
6.4.4 Displaying HTML with the JEditorPane ............. 494
6.5 Progress Indicators .............................................................................. 501
6.5.1 Progress Bars ............................................................................ 501
6.5.2 Progress Monitors .................................................................... 505
6.5.3 Monitoring the Progress of Input Streams ..................... 509
6.6 Component Organizers and Decorators .............................................. 514
6.6.1 Split Panes ............................................................................. 514
6.6.2 Tabbed Panes ........................................................................ 518
6.6.3 Desktop Panes and Internal Frames ................................... 524
6.6.4 Cascading and Tiling .............................................................. 527
6.6.5 Vetoing Property Settings ......................................................... 531
Chapter 7: Advanced AWT ................................................................. 549

7.1 The Rendering Pipeline .............................................................. 550
7.2 Shapes ......................................................................................... 553
  7.2.1 Using the Shape Classes .......................................................... 555
7.3 Areas ......................................................................................... 570
7.4 Strokes ......................................................................................... 572
7.5 Paint ......................................................................................... 581
7.6 Coordinate Transformations ....................................................... 583
7.7 Clipping ....................................................................................... 589
7.8 Transparency and Composition ................................................ 592
7.9 Rendering Hints ......................................................................... 601
7.10 Readers and Writers for Images ................................................. 608
  7.10.1 Obtaining Readers and Writers for Image File Types .......... 608
  7.10.2 Reading and Writing Files with Multiple Images ............... 610
7.11 Image Manipulation ................................................................. 619
  7.11.1 Constructing Raster Images .................................................... 619
  7.11.2 Filtering Images ................................................................. 626
7.12 Printing ...................................................................................... 636
  7.12.1 Graphics Printing ................................................................. 637
  7.12.2 Multiple-Page Printing ......................................................... 647
  7.12.3 Print Preview ........................................................................ 649
  7.12.4 Print Services ....................................................................... 659
  7.12.5 Stream Print Services ........................................................... 664
  7.12.6 Printing Attributes .............................................................. 664
7.13 The Clipboard ........................................................................... 672
  7.13.1 Classes and Interfaces for Data Transfer ......................... 674
  7.13.2 Transferring Text ................................................................. 674
  7.13.3 The Transferable Interface and Data Flavors ................. 678
  7.13.4 Building an Image Transferable ....................................... 680
  7.13.5 Transferring Java Objects via the System Clipboard ....... 685
  7.13.6 Using a Local Clipboard to Transfer Object References .... 689
7.14 Drag and Drop ......................................................................... 689
7.14.1 Data Transfer Support in Swing .............................................. 691
7.14.2 Drag Sources ............................................................................... 696
7.14.3 Drop Targets ............................................................................... 699
7.15 Platform Integration ................................................................................ 707
7.15.1 Splash Screens ............................................................................ 708
7.15.2 Launching Desktop Applications ............................................ 713
7.15.3 The System Tray ........................................................................ 719

Chapter 8: JavaBeans Components ......................................................... 725
8.1 Why Beans? .............................................................................................. 726
8.2 The Bean-Writing Process ...................................................................... 728
8.3 Using Beans to Build an Application ........................................... 731
8.3.1 Packaging Beans in JAR Files ................................................... 731
8.3.2 Composing Beans in a Builder Environment ........................ 733
8.4 Naming Patterns for Bean Properties and Events .............................. 740
8.5 Bean Property Types ............................................................................... 743
8.5.1 Simple Properties ....................................................................... 744
8.5.2 Indexed Properties .................................................................... 744
8.5.3 Bound Properties ....................................................................... 745
8.5.4 Constrained Properties ............................................................. 746
8.6 BeanInfo Classes .......................................................................................... 754
8.7 Property Editors .............................................................................................. 758
8.7.1 Writing Property Editors .......................................................... 762
8.7.1.1 String-Based Property Editors ............................................. 762
8.7.1.2 GUI-Based Property Editors .............................................. 765
8.8 Customizers ................................................................................................ 770
8.8.1 Writing a Customizer Class ..................................................... 772
8.9 JavaBeans Persistence ............................................................................. 779
8.9.1 Using JavaBeans Persistence for Arbitrary Data ................... 784
8.9.1.1 Writing a Persistence Delegate to Construct an Object .......... 784
8.9.1.2 Constructing an Object from Properties ......................... 786
8.9.1.3 Constructing an Object with a Factory Method .... 787
8.9.1.4 Postconstruction Work ....................................................... 787
8.9.1.5 Transient Properties ............................................................ 788
8.9.2 A Complete Example for JavaBeans Persistence .............. 791
Chapter 9: Security ................................................................................................. 803
9.1 Class Loaders ........................................................................................................ 804
  9.1.1 The Class Loader Hierarchy ................................................................. 806
  9.1.2 Using Class Loaders as Namespaces .................................................. 808
  9.1.3 Writing Your Own Class Loader ......................................................... 808
9.2 Bytecode Verification ......................................................................................... 816
9.3 Security Managers and Permissions ................................................................... 821
  9.3.1 Java Platform Security ........................................................................... 822
  9.3.2 Security Policy Files ............................................................................... 826
  9.3.3 Custom Permissions ............................................................................... 834
  9.3.4 Implementation of a Permission Class .................................................. 835
9.4 User Authentication ............................................................................................. 842
  9.4.1 JAAS Login Modules .............................................................................. 849
9.5 Digital Signatures ............................................................................................... 858
  9.5.1 Message Dests ....................................................................................... 859
  9.5.2 Message Signing ..................................................................................... 862
  9.5.3 Verifying a Signature .............................................................................. 865
  9.5.4 The Authentication Problem ................................................................. 868
  9.5.5 Certificate Signing .................................................................................. 870
  9.5.6 Certificate Requests ............................................................................... 872
9.6 Code Signing ....................................................................................................... 873
  9.6.1 JAR File Signing ..................................................................................... 873
  9.6.2 Software Developer Certificates ............................................................ 878
9.7 Encryption ........................................................................................................... 880
  9.7.1 Symmetric Ciphers ............................................................................... 881
  9.7.2 Key Generation ....................................................................................... 882
  9.7.3 Cipher Streams ....................................................................................... 887
  9.7.4 Public Key Ciphers ............................................................................... 888

Chapter 10: Scripting, Compiling, and Annotation Processing ....................... 893
10.1 Scripting for the Java Platform .......................................................................... 894
  10.1.1 Getting a Scripting Engine ..................................................................... 894
  10.1.2 Script Evaluation and Bindings ............................................................... 895
  10.1.3 Redirecting Input and Output ................................................................. 898
  10.1.4 Calling Scripting Functions and Methods .............................................. 899
  10.1.5 Compiling a Script ................................................................................. 901
10.1.6 An Example: Scripting GUI Events ................................................. 901
10.2 The Compiler API ................................................................................... 907
  10.2.1 Compiling the Easy Way .......................................................... 907
  10.2.2 Using Compilation Tasks ......................................................... 907
  10.2.3 An Example: Dynamic Java Code Generation ...................... 913
10.3 Using Annotations .................................................................................. 919
  10.3.1 An Example: Annotating Event Handlers ............................. 920
10.4 Annotation Syntax .................................................................................. 926
10.5 Standard Annotations ............................................................................. 931
  10.5.1 Annotations for Compilation ................................................... 932
  10.5.2 Annotations for Managing Resources .................................... 932
  10.5.3 Meta-Annotations ...................................................................... 933
10.6 Source-Level Annotation Processing ................................................... 935
10.7 Bytecode Engineering ............................................................................. 943
  10.7.1 Modifying Bytecodes at Load Time ........................................ 949

Chapter 11: Distributed Objects ....................................................................... 953
  11.1 The Roles of Client and Server .............................................................. 954
  11.2 Remote Method Calls ........................................................................ 957
    11.2.1 Stubs and Parameter Marshalling ........................................... 957
  11.3 The RMI Programming Model .............................................................. 959
    11.3.1 Interfaces and Implementations .............................................. 959
    11.3.2 The RMI Registry ................................................................ 961
    11.3.3 Deploying the Program ........................................................... 965
    11.3.4 Logging RMI Activity ............................................................. 968
  11.4 Parameters and Return Values in Remote Methods ................................ 970
    11.4.1 Transferring Remote Objects ................................................ 971
    11.4.2 Transferring Nonremote Objects ......................................... 971
    11.4.3 Dynamic Class Loading ....................................................... 974
    11.4.4 Remote References with Multiple Interfaces ......................... 979
    11.4.5 Remote Objects and the equals, hashCode, and clone Methods ........................................ 980
  11.5 Remote Object Activation ...................................................................... 980

Chapter 12: Native Methods .......................................................................... 989
  12.1 Calling a C Function from a Java Program ......................................... 990
  12.2 Numeric Parameters and Return Values ........................................... 997
12.2.1 Using printf for Formatting Numbers ........................................ 997
12.3 String Parameters ........................................................................ 999
12.4 Accessing Fields ........................................................................ 1005
  12.4.1 Accessing Instance Fields .................................................... 1005
  12.4.2 Accessing Static Fields ....................................................... 1009
12.5 Encoding Signatures .................................................................. 1010
12.6 Calling Java Methods ................................................................ 1012
  12.6.1 Instance Methods ............................................................... 1012
  12.6.2 Static Methods ................................................................. 1016
  12.6.3 Constructors ................................................................... 1017
  12.6.4 Alternative Method Invocations ....................................... 1018
12.7 Accessing Array Elements ........................................................... 1019
12.8 Handling Errors .......................................................................... 1023
12.9 Using the Invocation API .............................................................. 1028
12.10 A Complete Example: Accessing the Windows Registry ............. 1034
  12.10.1 Overview of the Windows Registry ..................................... 1034
  12.10.2 A Java Platform Interface for Accessing the Registry .......... 1036
  12.10.3 Implementation of Registry Access Functions as Native Methods ........................................ 1036

Index ............................................................................................................. 1051
To the Reader

The book you have in your hands is the second volume of the ninth edition of Core Java®, fully updated for Java SE 7. 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 on real projects.

Please note: If you are an experienced developer who is comfortable with advanced language features such as inner classes and generics, you need not have read the first volume in order to benefit from this volume. While we do refer to sections of the previous volume when appropriate (and, of course, hope you will buy or have bought Volume I), you can find all the background material you need in any comprehensive introductory book about the Java platform.

Finally, 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.

The topic of Chapter 1 is input and output handling (I/O). In Java, all I/O is handled through so-called streams. Streams 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 the NIO2 library of Java SE 7, which makes common operations (such as reading all lines in a file) very convenient.

Chapter 2 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 3 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 4 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 long 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).

Chapter 5 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 not only cross platforms but cross country boundaries as well. For example, we show you how to write a retirement calculator that uses either English, German, or Chinese languages.

Chapter 6 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 7 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 8 explains what you need to know about the component API for the Java platform—JavaBeans. We show you how to write your own beans that
other programmers can manipulate in integrated builder environments. We conclude this chapter by showing you how you can use JavaBeans persistence to store your data in a format that—unlike object serialization—is suitable for long-term storage.

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 covers distributed objects. We cover RMI (Remote Method Invocation) in detail. This API lets you work with Java objects that are distributed over multiple machines.

Chapter 11 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 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 the Java platform 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 7 are covered in detail.

Conventions

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

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.
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 (Principal Consultant, Quiotix Corp.), Angela Gordon, Dan Gordon, Rob Gordon, John Gray (University of Hartford), Cameron Gregory (olabs.com), 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 (Channel-Point 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), 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), 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), Peter van der Linden, Burt Walsh, Joe Wang (Oracle), and Dan Xu (Oracle).

Cay Horstmann
San Francisco, California
December 2012
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.
10.1 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. (The other direction—accessing Java from the scripting language—is the responsibility of the scripting language provider. Most scripting languages that run on the Java virtual machine have this capability.)

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

10.1.1 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 10.1 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("JavaScript");
```

Java SE 7 includes a version of Rhino, a JavaScript interpreter developed by the Mozilla foundation. You can add more languages by providing the necessary JAR files on the class path. You will generally need two sets of JAR files. The scripting language itself is implemented by a single JAR file or a set of JARs. The engine that adapts the language to the scripting API usually requires an additional JAR. The site http://java.net/projects/scripting provides engines for a
Table 10.1 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>Rhino (included with Java SE)</td>
<td>js, rhino, 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>SISC Scheme</td>
<td>scheme, sisc</td>
<td>None</td>
<td>scc, sce, scm, shp</td>
</tr>
</tbody>
</table>

A wide range of scripting languages. For example, to add support for Groovy, the class path should contain `groovy/lib/*` (from http://groovy.codehaus.org) and `groovy-engine.jar` (from http://java.net/projects/scripting).

```java
javax.script.ScriptEngineManager 6

* 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.
```

```java
javax.script.ScriptEngineFactory 6

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

### 10.1.2 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'");
```

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

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

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:
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.

### javax.script.Bindings

- `Object get(String key)`
- `void put(String key, Object value)`
gets or puts a binding into the scope represented by this `Bindings` object.
10.1.3 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.

**CAUTION:** You can pass any `Writer` to the `setWriter` method, but the Rhino engine throws an exception if it is not a `PrintWriter`.

The `setReader` and `setWriter` methods only affect the scripting engine’s standard input and output sources. For example, if you execute the JavaScript code

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

only the first output is redirected.

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

### `javax.script.ScriptEngine`

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ScriptContext getContext()</code></td>
</tr>
</tbody>
</table>
| gets the default script context for this engine.

### `javax.script.ScriptContext`
10.1.4 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 Rhino engine implements `Invocable`.

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

```java
if (engine implements Invocable)
    ((Invocable) engine).invokeFunction("aFunction", param1, param2);
```

If the scripting language is object-oriented, you call a method like this:

```java
((Invocable) engine).invokeMethod(implicitParam, "aMethod", explicitParam1, explicitParam2);
```

Here, the `implicitParam` object is a proxy to an object in the scripting language. It must be the result of a prior call to the scripting engine.

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 `getMethodImplSyntax` 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 greet(String whom);
}
```

In Rhino, you provide a function

```java
function greet(x) { return "Hello, " + x + "!"; }
```

This code must be evaluated first. Then you can call

```java
Greeter g = ((Invocable) engine).getInterface(Greeter.class);
```
Now you can make a plain Java method call

    String result = g.greet("World");

Behind the scenes, the JavaScript `greet` method is invoked. This approach is similar to making a remote method call, as discussed in Chapter 11.

In an object-oriented scripting language, you can access a script class through a matching Java interface. For example, consider this JavaScript code, which defines a `SimpleGreeter` class.

    function SimpleGreeter(salutation) { this.salutation = salutation; }
    SimpleGreeter.prototype.greet = function(whom) { return this.salutation + ", " + whom + ";!"; }

You can use this class to construct greeters with different salutations (such as "Hello", "Goodbye", and so on).

**NOTE:** For more information on how to define classes in JavaScript, see *JavaScript—The Definitive Guide, Fifth Edition*, by David Flanagan (O'Reilly, 2006).

After evaluating the JavaScript class definition, call

    Object goodbyeGreeter = engine.eval("new SimpleGreeter('Goodbye')");
    Greeter g = ((Invocable) engine).getInterface(goodbyeGreeter, Greeter.class);

When you call `g.greet("World")`, the `greet` method is invoked on the JavaScript object `goodbyeGreeter`. The result is a string "Goodbye, 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** 6

- `Object invokeFunction(String name, Object... parameters)`
  - 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.
10.1.5 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)
    CompiledScript 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.

<table>
<thead>
<tr>
<th>javax.script.Compilable 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>- CompiledScript compile(String script)</td>
</tr>
<tr>
<td>- CompiledScript compile(Reader reader)</td>
</tr>
</tbody>
</table>

compiles the script given by a string or reader.

<table>
<thead>
<tr>
<th>javax.script.CompiledScript 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Object eval()</td>
</tr>
<tr>
<td>- Object eval(Bindings bindings)</td>
</tr>
</tbody>
</table>

evaluates this script.

10.1.6 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 10.1 that adds scripting to an arbitrary frame class. By default it reads the ButtonFrame class in Listing 10.2, 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:

```plaintext
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 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 seems like a good idea in general; moreover, the Scheme interpreter needs some cumbersome 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 the engine scope.

Then 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, together with the section on JavaBeans events in Chapter 8 of this volume, if you want to follow the implementation in detail. The essential part, however, is that each event handler calls

```java
engine.eval(scriptCode);
```

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

```plaintext
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

```java
panel.background = java.awt.Color.YELLOW
```

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
java -classpath .:groovy/lib/*:jsr223-engines/groovy/build/groovy-engine.jar ScriptTest groovy
```

Here, `groovy` is the directory into which you installed Groovy, and `jsr223-engines` is the directory that contains the engine adapters from http://java.net/projects/scripting.

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

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

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 2. 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.

```
Listing 10.1  script/ScriptTest.java

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 1.01 2012-01-28
 * @author Cay Horstmann
 */
public class ScriptTest
{
   public static void main(final String[] args)
   {
      EventQueue.invokeLater(new Runnable()
      {
         public void run()
         {
            try
            {
               ScriptEngineManager manager = new ScriptEngineManager();
               (Continues)
```
String language;
if (args.length == 0)
{
    System.out.println("Available factories: ");
    for (ScriptEngineFactory factory : manager.getEngineFactories())
        System.out.println(factory.getEngineName());

    language = "js";
}
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));
getComponentBindings(frame, engine);

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);
}
frame.setTitle("ScriptTest");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
/**
* Gathers all named components in a container.
* @param c the component
* @param namedComponents
*/
private static void getComponentBindings(Component c, ScriptEngine engine)
{
    String name = c.getName();
    if (name != null) engine.put(name, c);
    if (c instanceof Container)
    {
        for (Component child : ((Container) c).getComponents())
            getComponentBindings(child, engine);
    }
}

/**
* 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) throws ReflectiveOperationException, IntrospectionException
{
    Object bean = engine.get(beanName);
    EventSetDescriptor descriptor = getEventSetDescriptor(bean, eventName);
    if (descriptor == null) return;
    descriptor.getAddListenerMethod().invoke(bean,
        Proxy.newProxyInstance(null, new Class[] { descriptor.getListenerType() },
            new InvocationHandler()
            {
                public Object invoke(Object proxy, Method method, Object[] args)
                    throws Throwable
                {
                    engine.eval(scriptCode);
                    return null;
                }
            }));
}

private static EventSetDescriptor getEventSetDescriptor(Object bean, String eventName)
throws IntrospectionException
{
(Continues)
for (EventSetDescriptor descriptor : Introspector.getBeanInfo(bean.getClass()).getEventSetDescriptors())
    if (descriptor.getName().equals(eventName)) return descriptor;
return null;
}

### Listing 10.2  buttons1/ButtonFrame.java

```java
package buttons1;

import javax.swing.*;

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);
    }
}
```
10.2 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. As of Java SE 6, 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.

10.2.1 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.

10.2.2 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 10.3
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 ex-
ample 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.

```java
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 10.4 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:

```java
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.
package compiler;

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");
    }
}

Listing 10.4 compiler/ByteArrayJavaClass.java

package compiler;

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();
    }
}

**javax.tools.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.

**javax.tools.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.
- String getMessage(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.
javax.tools.SimpleJavaFileObject

- 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.

javax.tools.ForwardingJavaFileManager<

- 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.

10.2.3 An Example: Dynamic Java Code Generation

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

```html
<p>The current date and time is <%= new java.util.Date() %></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 10.5 shows a very simple example of a frame class, and Listing 10.6 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 x, which we hope is not used anywhere else in the program. The generated code has the form
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 handler1
            }
        });
        // repeat for the other event handlers ...
    }
}

The buildSource method in the program of Listing 10.7 builds up this code and places it into a StringBuilderJavaSource object. That object is passed to the Java compiler.

We use a ForwardingJavaFileManager with a getJavaFileForOutput method that constructs a ByteArrayJavaClass object for every class in the x package. These objects capture the class files generated when the 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 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 10.8) loads the classes stored in this map.

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

    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.
package buttons2;
import javax.swing.*;

/**
   * @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();
    }
}

Listing 10.6 buttons2/action.properties

yellowButton=panel.setBackground(java.awt.Color.YELLOW);
blueButton=panel.setBackground(java.awt.Color.BLUE);
Listing 10.7  compiler/CompilerTest.java

```java
package compiler;

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

/**
 * @version 1.00 2007-10-28
 * @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(new Runnable()
{
    public void run()
    {
        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;\n\n");
    source.append("public class Frame extends " + superclassName + " {\n");
    final Properties props = new Properties();
    props.load(Class.forName(superclassName).getResourceAsStream("action.properties"));
    (Continues)
for (Map.Entry<Object, Object> e : props.entrySet())
    {
        String beanName = (String) e.getKey();
        String eventCode = (String) e.getValue();
        source.append(beanName + ".addActionListener(new java.awt.event.ActionListener() {");
        source.append("public void actionPerformed(java.awt.event.ActionEvent event) {");
        source.append(eventCode);
        source.append("} } } });");
        source.append("} } ");
        return source;
    }
}

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<?> cl = defineClass(name, classBytes, 0, classBytes.length);
        if (cl == null) throw new ClassNotFoundException(name);
        return cl;
    }
}
10.3 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. You need to 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

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

Annotations can be defined to have \textit{elements}, such as

\begin{verbatim}
@Test(timeout="10000")
\end{verbatim}

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 \texttt{public} or \texttt{static}.

Each annotation must be defined by an \textit{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:

\begin{verbatim}
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Test {
    long timeout() default 0L;
    ...
}
\end{verbatim}

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

The \texttt{Target} and \texttt{Retention} annotations are \textit{meta-annotations}. They annotate the \texttt{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 10.5.3, “Meta-Annotations,” on p. 933.

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.

### 10.3.1 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
myButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        doSomething();
    }
});

In this section, we’ll design an annotation to avoid this drudgery. The annotation, defined in Listing 10.9, is used as follows:

    @ActionListenerFor(source="myButton") void doSomething() {

The programmer no longer has to make calls to addActionListener. Instead, each method is simply tagged with an annotation. Listing 10.10 shows the ButtonFrame class from Volume I, Chapter 8, reimplemented with these annotations.

We also need to define an annotation interface. The code is in Listing 10.11.

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

    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.

    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.

    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 10.1 shows how annotations are handled in this example.

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 10.7, “Bytecode Engineering,” on p. 943.

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.
Listing 10.9 runtimeAnnotations/ActionListenerInstaller.java

```java
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
     */
```
Listing 10.9  (Continued)

```java
44     public static void addListener(Object source, final Object param, final Method m)
45        throws ReflectiveOperationException
46     {
47        InvocationHandler handler = new InvocationHandler()
48           {
49              public Object invoke(Object proxy, Method mm, Object[] args) throws Throwable
50              {
51                  return m.invoke(param);
52              }
53           };
54
55        Object listener = Proxy.newProxyInstance(null,
56              new Class[] { java.awt.event.ActionListener.class }, handler);
57        Method adder = source.getClass().getMethod("addActionListener", ActionListener.class);
58        adder.invoke(source, listener);
59     }
```

Listing 10.10  buttons3/ButtonFrame.java

```java
1  package buttons3;
2
3  import java.awt.*;
4  import javax.swing.*;
5  import runtimeAnnotations.*;
6
7  /**
8   * A frame with a button panel.
9   * @version 1.00 2004-08-17
10   * @author Cay Horstmann
11   */
12  public class ButtonFrame extends JFrame
13  {
14     private static final int DEFAULT_WIDTH = 300;
15     private static final int DEFAULT_HEIGHT = 200;
16
17     private JPanel panel;
18     private JButton yellowButton;
19     private JButton blueButton;
20     private JButton redButton;
21
22     public ButtonFrame()
23     {
24         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
25         panel = new JPanel();
26         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);
}
10.4 Annotation Syntax

In this section, we cover everything you need to know about the annotation syntax.

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`:

```java
public @interface BugReport
{
    String assignedTo() default "[none]";
    int severity() = 0;
}
```

Each annotation has the format

```java
@AnnotationName(elementName1=value1, elementName2=value2, ...)
```
For example,

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

The order of the elements does not matter. The annotation

```java
@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

```java
@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

@ActionListenerFor(value="yellowButton")

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. Instead, the virtual machine generates proxy classes and objects when needed. For example, when requesting an ActionListenerFor annotation, the virtual machine carries out an operation similar to the following:

```java
return Proxy.newProxyInstance(classLoader, ActionListenerFor.class, new InvocationHandler()
{
    public Object invoke(Object proxy, Method m, Object[] args) throws Throwable
    {
        if (m.getName().equals("source")) return value of source annotation;
    ... 
});
```

The element declarations in the annotation interface are actually method declarations. The methods of an annotation interface can have no parameters and no throws clauses, and they cannot be generic.

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 of the preceding types (an array of arrays is not a legal element 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();
}

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`.

You can add annotations to the following items:

- Packages
- Classes (including `enum`)
- Interfaces (including annotation interfaces)
- Methods
- Constructors
- Instance fields (including `enum` constants)
- Local variables
- Parameter variables
However, 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.

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

An item can have multiple annotations, provided they belong to different types. You cannot use the same annotation type more than once when annotating a particular item. For example,

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

is a compile-time error. If this is a problem, design an annotation whose value is an array of simpler annotations:

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

---

<table>
<thead>
<tr>
<th><code>java.lang.annotation.Annotation</code> 5.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Class&lt;? extends Annotation&gt; annotationType() returns the Class object that represents the annotation interface of this annotation object. Note that calling <code>getClass</code> on an annotation object would return the actual class, not the interface.</td>
</tr>
<tr>
<td>• boolean equals(Object other) returns <code>true</code> if <code>other</code> is an object that implements the same annotation interface as this annotation object and if all elements of this object and <code>other</code> are equal.</td>
</tr>
<tr>
<td>• int hashCode() returns a hash code, compatible with the <code>equals</code> method, derived from the name of the annotation interface and the element values.</td>
</tr>
<tr>
<td>• String toString() returns a string representation that contains the annotation interface name and the element values; for example, <code>@BugReport(assignedTo=[none], severity=0)</code>.</td>
</tr>
</tbody>
</table>
10.5 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 10.2 shows these annotations. We’ll discuss them in detail in the following two sections.

Table 10.2 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>Override</td>
<td>Methods</td>
<td>Checks that this method overrides a superclass 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>Methods</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>
<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>
</tbody>
</table>
10.5.1 Annotations for Compilation

The `@Deprecated` annotation can be attached to any items for which 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");
```

10.5.2 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:
@Resource(name="jdbc/mydb")
private DataSource source;

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

10.5.3 Meta-Annotations

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

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

Table 10.3 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 10.4. The default is RetentionPolicy.CLASS.

In Listing 10.11 on p. 925, 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.

**Table 10.3 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>
</tbody>
</table>
Table 10.4 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>

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 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 10.2.

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 `@Persistent` to indicate that objects of a class can be saved in a database. Then the subclasses of persistent classes are automatically annotated as persistent.

```java
@Inherited @interface Persistent { }
@Persistent class Employee { . . . }
class Manager extends Employee { . . . } // also @Persistent
```

When the persistence mechanism searches for objects to store in the database, it will detect both `Employee` and `Manager` objects.

### 10.6 Source-Level Annotation Processing

One use for annotation is the automatic generation of “side files” that contain additional information about programs. In the past, the Enterprise Edition of Java was notorious for making programmers fuss with lots of boilerplate code. Modern versions of Java EE use annotations to greatly simplify the programming model.

In this section, we demonstrate this technique with a simpler example. We write a program that automatically produces bean info classes. You tag bean properties
with an annotation and then run a tool that parses the source file, analyzes the annotations, and writes out the source file of the bean info class.

Recall from Chapter 8 that a bean info class describes a bean more precisely than the automatic introspection process can. The bean info class lists all of the properties of the bean. Properties can have optional property editors. The ChartBeanBeanInfo class in Chapter 8 is a typical example.

To eliminate the drudgery of writing bean info classes, we supply an @Property annotation. You can tag either the property getter or setter, like this:

```java
@Property String getTitle() { return title; }
```

or

```java
@Property(editor="TitlePositionEditor")
public void setTitlePosition(int p) { titlePosition = p; }
```

Listing 10.12 contains the definition of the @Property annotation. Note that the annotation has a retention policy of SOURCE. We analyze the annotation at the source level only. It is not included in class files and not available during reflection.

### Listing 10.12  sourceAnnotations/Property.java

```java
package sourceAnnotations;
import java.lang.annotation.*;

@Documented
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.SOURCE)
public @interface Property
{
    String editor() default "";
}
```

**NOTE:** It would have made sense to declare the editor element to have type Class. However, the annotation processor cannot retrieve annotations of type Class because the meaning of a class can depend on external factors (such as the class path or class loaders). Therefore, we use a string to specify the editor class name.

To automatically generate the bean info class of a class with name BeanClass, we carry out the following tasks:

1. Write a source file BeanClassBeanInfo.java. Declare the BeanClassBeanInfo class to extend SimpleBeanInfo, and override the getPropertyDescriptors method.
2. For each annotated method, recover the property name by stripping off the `get` or `set` prefix and “decapitalizing” the remainder.
3. For each property, write a statement for constructing a `PropertyDescriptor`.
4. If the property has an editor, write a method call to `setPropertyEditorClass`.
5. Write the code for returning an array of all property descriptors.

For example, the annotation

```java
@Property(editor="TitlePositionEditor")
public void setTitlePosition(int p) { titlePosition = p; }
```

in the `ChartBean` class is translated into

```java
public class ChartBeanBeanInfo extends java.beans.SimpleBeanInfo
{
    public java.beans.PropertyDescriptor[] getProperties()
    {
        java.beans.PropertyDescriptor titlePositionDescriptor
            = new java.beans.PropertyDescriptor("titlePosition", ChartBean.class);
        titlePositionDescriptor.setPropertyEditorClass(TitlePositionEditor.class)
        . . .
        return new java.beans.PropertyDescriptor[]
            {
            titlePositionDescriptor,
                . . .
            }
    }
}
```

(The boilerplate code is printed in the lighter gray.)

All this is easy enough to do, provided we can locate all methods that have been tagged with the `@Property` annotation.

As of Java SE 6, you can add *annotation processors* to the Java compiler. (In Java SE 5, a stand-alone tool, called apt, was used for the same purpose.) To invoke annotation processing, run

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

The compiler locates the annotations of the source files. It then selects the annotation processors that should be applied. Each annotation processor is executed in turn. If an annotation processor creates a new source file, then the process is repeated. Once a processing round yields no further source files, all source files are compiled. Figure 10.3 shows how the `@Property` annotations are processed.

We do not discuss the annotation processing API in detail, but the program in Listing 10.13 will give you a flavor of its capabilities.
An annotation processor implements the `Processor` interface, generally by extending the `AbstractProcessor` class. You need to specify which annotations your processor supports. The designers of the API themselves love annotations, so they use an annotation for this purpose:

```java
@SupportedAnnotationTypes("com.horstmann.annotations.Property")
public class BeanInfoAnnotationProcessor extends AbstractProcessor
```

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 `BeanInfoAnnotationProcessor` has a single public method, `process`, that is called for each file. The `process` method has two parameters: the set of annotations that is being processed in this round, and a `RoundEnv` reference that contains information about the current processing round.

In the `process` method, we iterate through all annotated methods. For each method, we get the property name by stripping off the `get`, `set`, or `is` prefix and changing the next letter to lower case. Here is the outline of the code:

```java
public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv)
{
    for (TypeElement t : annotations)
    {
        Map<String, Property> props = new LinkedHashMap<String, Property>();
        for (Element e : roundEnv.getElementsAnnotatedWith(t))
        {
            props.put(property name, e.getAnnotation(Property.class));
        }
        write bean info source file
        return true;
    }
}
```

The `process` method should return `true` if it claims all the annotations presented to it; that is, if those annotations should not be passed on to other processors.

The code for writing the source file is straightforward—just a sequence of `out.print` statements. Note that we create the output writer as follows:

```java
JavaFileObject sourceFile = processingEnv.getFiler().createSourceFile(beanClassName + "BeanInfo");
PrintWriter out = new PrintWriter(sourceFile.openWriter());
```

The `AbstractProcessor` class has a protected field `processingEnv` for accessing various processing services. The `Filer` interface is responsible for creating new files and tracking them so that they can be processed in subsequent processing rounds.
When an annotation processor detects an error, it uses the `Messager` to communicate with the user. For example, we issue an error message if a method has been annotated with `@Property` but its name doesn’t start with `get`, `set`, or `is`:

```java
if (!found) processingEnv.getMessager().printMessage(Kind.ERROR, 
    "@Property must be applied to getXxx, setXxx, or isXxx method", e);
```

In the companion code for this book, we supply an annotated file, `ChartBean.java`. Compile the annotation processor:

```
javac sourceAnnotations/BeanInfoAnnotationProcessor.java
```

Then run

```
javac -processor sourceAnnotations.BeanInfoAnnotationProcessor chart/ChartBean.java
```

and have a look at the automatically generated file `ChartBeanBeanInfo.java`.

To see the annotation processing in action, add the command-line option `XprintRounds` to the `javac` command. You will get this output:

```
Round 1:
  input files: {com.horstmann.corejava.ChartBean}
  annotations: [com.horstmann.annotations.Property]
  last round: false
Round 2:
  input files: {com.horstmann.corejava.ChartBeanBeanInfo}
  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

```java
@Property private String title;
```

could produce the methods

```java
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.

---

**Listing 10.13** sourceAnnotations/BeanInfoAnnotationProcessor.java

```java
package sourceAnnotations;

import java.beans.*;
import java.io.*;
import java.util.*;
import javax.annotation.processing.*;
import javax.lang.model.*;
import javax.lang.model.element.*;
import javax.tools.*;
import javax.tools.Diagnostic.*;

/**
 * This class is the processor that analyzes Property annotations.
 * @version 1.11 2012-01-26
 * @author Cay Horstmann
 */
@SupportedAnnotationTypes("sourceAnnotations.Property")
@SupportedSourceVersion(SourceVersion.RELEASE_7)
public class BeanInfoAnnotationProcessor extends AbstractProcessor {

```
```java
@Override
public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv)
{
    for (TypeElement t : annotations)
    {
        Map<String, Property> props = new LinkedHashMap<>();
        String beanClassName = null;
        for (Element e : roundEnv.getElementsAnnotatedWith(t))
        {
            String mname = e.getSimpleName().toString();
            String[] prefixes = { "get", "set", "is" };
            boolean found = false;
            for (int i = 0; !found && i < prefixes.length; i++)
                if (mname.startsWith(prefixes[i]))
                {
                    found = true;
                    int start = prefixes[i].length();
                    String name = Introspector.decapitalize(mname.substring(start));
                    props.put(name, e.getAnnotation(Property.class));
                }
            if (!found) processingEnv.getMessager().printMessage(Kind.ERROR,
                "@Property must be applied to getXxx, setXxx, or isXxx method", e);
            else if (beanClassName == null)
                beanClassName = ((TypeElement) e.getEnclosingElement()).getQualifiedName()
                .toString();
        }
        try
        {
            if (beanClassName != null) writeBeanInfoFile(beanClassName, props);
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
    return true;
}

/**
 * Writes the source file for the BeanInfo class.
 * @param beanClassName the name of the bean class
 * @param props a map of property names and their annotations
 */
private void writeBeanInfoFile(String beanClassName, Map<String, Property> props)
    throws IOException
{
    JavaFileObject sourceFile = processingEnv.getFiler().createSourceFile(
        beanClassName + "BeanInfo" motivations).
    (Continues)
```
Listing 10.13 (Continued)

70        PrintWriter out = new PrintWriter(sourceFile.openWriter());
71        int i = beanClassName.lastIndexOf(".");
72        if (i > 0)
73        {
74           out.print("package ");
75           out.print(beanClassName.substring(0, i));
76           out.println(";");
77        }
78        out.print("public class ");
79        out.print(beanClassName.substring(i + 1));
80        out.println("BeanInfo extends java.beans.SimpleBeanInfo");
81        out.println("{\n");
82        out.println("   public java.beans.PropertyDescriptor[] getPropertyDescriptors()\n");
83        out.println("   {\n");
84        out.println("      try\n");
85        out.println("      {\n");
86        for (Map.Entry<String, Property> e : props.entrySet())
87        {
88           out.print("         java.beans.PropertyDescriptor "\n");
89           out.print(e.getKey());
90           out.println("Descriptor\n");
91           out.print("            = new java.beans.PropertyDescriptor("\n");
92           out.print(e.getKey());
93           out.println("", ");
94           out.print(beanClassName);
95           out.println(".class;">;
96           String ed = e.getValue().editor().toString();
97           if (!ed.equals(""))
98           {
99              out.print("         Description \n");
100             out.print(e.getKey());
101             out.println("Descriptor.setEditorClass("\n");
102             out.print(ed);
103             out.println(".class;">;
104          }
105       }
106       out.println("         return new java.beans.PropertyDescriptor[]\n");
107       out.println("      \n");
108       boolean first = true;
109       for (String p : props.keySet())
110       {
111          if (first) first = false;
112          else out.print(",");
113          out.println("\n");
114          out.print("      \n");
115          out.print(p);
116          out.println("Descriptor\n");
117       }
118       out.println();
119       out.println("  ");
120       out.println("  ");
121       out.println(" catch (java.beans.IntrospectionException e)");
122       out.println("  ");
123       out.println(" e.printStackTrace();");
124       out.println(" return null;");
125       out.println("  ");
126       out.println(" ");
127       out.println(" ");
128       out.close();
129    }
130 }

10.7 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/se7/html). The format is rather complex, and it would be challenging to process class files without special libraries. One such library is the Bytecode Engineering Library (BCEL), available at http://jakarta.apache.org/bcel.

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

\@LogEntry(logger=loggerName)

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

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

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

\@LogEntry(logger="global") public int hashCode()

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

Aug 17, 2004 9:32:59 PM 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:

```
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 BCEL 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 10.14 edits a class file and inserts a logging call at the beginning of the methods annotated with the LogEntry annotation.

**NOTE:** If you are interested in the details of bytecode engineering, read through the BCEL manual at http://jakarta.apache.org/bcel/manual.html.

You’ll need version 6.0 or later of the BCEL library to compile and run the EntryLogger program. (As this chapter was written, that version was still a work in progress. If it isn’t finished when you read this, check out the trunk from the Subversion repository.)

For example, here is how you add the logging instructions to Item.java in Listing 10.15:

```
javac set/Item.java
javac -classpath .:bcel-version.jar bytecodeAnnotations/EntryLogger.java
java -classpath .:bcel-version.jar 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.

```java
public int hashCode();
Code:
0: ldc    #85; //String global
2: invokevirtual java/util/logging/Logger.getLogger:(Ljava/lang/String;)Ljava/util/logging/Logger;
5: ldc    #86; //String Item
7: ldc    #88; //String hashCode
9: invokevirtual jav-util/logging/Logger.entering:(Ljava/lang/String;Ljava/lang/String;)V
12: bipush 13
14: aload_0
```
The SetTest program in Listing 10.16 inserts Item objects into a hash set. When you run it with the modified class file, you will see the logging messages.

Aug 18, 2004 10:57:59 AM Item hashCode
FINER: ENTRY
Aug 18, 2004 10:57:59 AM Item hashCode
FINER: ENTRY
Aug 18, 2004 10:57:59 AM Item hashCode
FINER: ENTRY
Aug 18, 2004 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 10.14  bytecodeAnnotations/EntryLogger.java

```java
package bytecodeAnnotations;
import java.io.*;
import org.apache.bcel.*;
import org.apache.bcel.classfile.*;
import org.apache.bcel.generic.*;

/**
 * Adds "entering" logs to all methods of a class that have the LogEntry annotation.
 * @version 1.10 2007-10-27
 * @author Cay Horstmann
 */
public class EntryLogger {
    private ClassGen cg;
    private ConstantPoolGen cpg;
}
(Continues)```
Listing 10.14  (Continued)

```java
    /**
     * Adds entry logging code to the given class.
     * @param args the name of the class file to patch
     */
    public static void main(String[] args)
    {
        try
        {
            if (args.length == 0)
                System.out.println("USAGE: java bytecodeAnnotations.EntryLogger classname");
            else
            {
                JavaClass jc = Repository.lookupClass(args[0]);
                ClassGen cg = new ClassGen(jc);
                EntryLogger el = new EntryLogger(cg);
                el.convert();
                String f = Repository.lookupClassFile(cg.getClassName()).getPath();
                System.out.println("Dumping " + f);
                cg.getJavaClass().dump(f);
            }
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    /**
     * Constructs an EntryLogger that inserts logging into annotated methods of a given class.
     * @param cg the class
     */
    public EntryLogger(ClassGen cg)
    {
        this.cg = cg;
        cpg = cg.getConstantPool();
    }

    /**
     * converts the class by inserting the logging calls.
     */
    public void convert() throws IOException
    {
        for (Method m : cg.getMethods())
        {
            AnnotationEntry[] annotations = m.getAnnotationEntries();
```
for (AnnotationEntry a : annotations)
{
    if (a.getAnnotationType().equals("LbytecodeAnnotations/LogEntry;"))
    {
        for (ElementValuePair p : a.getElementValuePairs())
        {
            if (p.getNameString().equals("logger"))
            {
                String loggerName = p.getValue().stringifyValue();
                cg.replaceMethod(m, insertLogEntry(m, loggerName));
            }
        }
    }
}

/**
 * Adds an "entering" call to the beginning of a method.
 * @param m the method
 * @param loggerName the name of the logger to call
 */
private Method insertLogEntry(Method m, String loggerName)
{
    MethodGen mg = new MethodGen(m, cg.getClassName(), cpg);
    String className = cg.getClassName();
    String methodName = mg.getMethod().getName();
    System.out.printf("Adding logging instructions to %s.%s%n", className, methodName);

    int getLoggerIndex = cpg.addMethodref("java.util.logging.Logger", "getLogger",
            "(Ljava/lang/String;)Ljava/util/logging/Logger;");
    int enteringIndex = cpg.addMethodref("java.util.logging.Logger", "entering",
            "(Ljava/lang/String;Ljava/lang/String;)V");

    InstructionList il = mg.getInstructionList();
    InstructionList patch = new InstructionList();
    patch.append(new PUSH(cpg, loggerName));
    patch.append(new INVOKESTATIC(getLoggerIndex));
    patch.append(new PUSH(cpg, className));
    patch.append(new PUSH(cpg, methodName));
    patch.append(new INVOKEVIRTUAL(enteringIndex));
    InstructionHandle[] ihs = il.getInstructionHandles();
    il.insert(ihs[0], patch);
    mg.setMaxStack();
    return mg.getMethod();
}
Listing 10.15  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 = "global")
    public boolean equals(Object otherObject)
    {
        if (this == otherObject) return true;
        if (otherObject == null) return false;
        if (getClass() != otherObject.getClass()) return false;
        if (getDescription().equals(otherObject.getDescription())) return true;
        return false;
    }
```
Item other = (Item) otherObject;
return Objects.equals(description, other.description) && partNumber == other.partNumber;
}

@LogEntry(logger = "global")
public int hashCode()
{
    return Objects.hash(description, partNumber);
}

Listing 10.16  set/SetTest.java

package set;
import java.util.*;
import java.util.logging.*;

/**
 * @version 1.02 2012-01-26
 * @author Cay Horstmann
 */
public class SetTest
{
    public static void main(String[] args)
    {
        Logger.getLogger(Logger.GLOBAL_LOGGER_NAME).setLevel(Level.FINEST);
        Handler handler = new ConsoleHandler();
        handler.setLevel(Level.FINEST);
        Logger.getLogger(Logger.GLOBAL_LOGGER_NAME).addHandler(handler);

        Set<Item> parts = new HashSet<>();
        parts.add(new Item("Toaster", 1279));
        parts.add(new Item("Microwave", 4104));
        parts.add(new Item("Toaster", 1279));
        System.out.println(parts);
    }
}

10.7.1 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.
Before Java SE 5.0, you had to write a custom class loader to achieve this task. Now, 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:
   ```
   Premain-Class: bytecodeAnnotations.EntryLoggingAgent
   ```

3. Package the agent code and the manifest into a JAR file, for example:
   ```
   javac -classpath .:bcel-version.jar bytecodeAnnotations.EntryLoggingAgent
   jar cvfm EntryLoggingAgent.jar 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 SetTest.java
   ```

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

Listing 10.17 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 10.4). In other words, this technique carries out “just in time” modification of the bytecodes.
Listing 10.17  bytecodeAnnotations/EntryLoggingAgent.java

```java
package bytecodeAnnotations;

import java.lang.instrument.*;
import java.io.*;
import java.security.*;
import org.apache.bcel.classfile.*;
import org.apache.bcel.generic.*;

/**
 * @version 1.00 2004-08-17
 * @author Cay Horstmann
 */
public class EntryLoggingAgent
{
    public static void premain(final String arg, Instrumentation instr)
    {
        instr.addTransformer(new ClassFileTransformer()
        {
            public byte[] transform(ClassLoader loader, String className, Class<?> cl,
                ProtectionDomain pd, byte[] data)
            {
                if (!className.equals(arg)) return null;
                try
                {
                    ClassParser parser = new ClassParser(new ByteArrayInputStream(data), className + ".java");
                    JavaClass jc = parser.parse();
                    ClassGen cg = new ClassGen(jc);
                    EntryLogger el = new EntryLogger(cg);
                    el.convert();
                    return cg.getJavaClass().getBytes();
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                    return null;
                }
            }
        });
    }
}
```
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, you will learn about the RMI mechanism, a distributed object model for Java programs.
Symbols and Numbers

- (minus sign)
  - in permissions, 832
  - in policy files, 829
  - in regular expressions, 82
  - in URLs, 224

-
  - (negative infinity), in message formatting, 339

_ (underscore)
  - in native method names, 991
  - in SQL, 243, 272
  - in URLs, 224

, (comma)
  - decimal, 306, 312, 468
  - in DTDs, 116

; (semicolon)
  - in classpath, 247
  - in method signatures, 1011
  - in SQL, 247
  - not needed, in annotations, 919

: (colon)
  - as delimiter in text files, 16
  - in classpath, 247
  - in permissions, 832
  - in URLs, 211

!= operator (SQL), 243

? (question mark)
  - for characters from missing fonts, 21, 321
  - in DTDs, 116
  - in e-mail URIs, 714
  - in glob patterns, 66
  - in masks, 474
  - in prepared queries, 263
  - in regular expressions, 82, 84
  - in URLs, 224

/ (slash)
  - ending, in codebase URLs, 828, 968
  - in method signatures, 1011
  - in path names, 57
  - in URLs, 211

. (period)
  - decimal, 306, 312, 468
  - in method signatures, 1011
  - in native method names, 991

  - in regular expressions, 82–83, 86
  - in URLs, 224
  - leading, in file names, 827

.., in paths, 58

^ (caret), in regular expressions, 82–83, 86

~ (tilde), in URLs, 224

' (single quote), in masks, 474

'..' (single quotes), in SQL, 243

"." (double quotes), in HTML vs. XML, 96

(…) (parentheses)
  - in method signatures, 1011
  - in regular expressions, 82, 84

[ (array), type code, 44, 1011

[ ] (square brackets)
  - in DOCTYPE declaration, 114
  - in glob patterns, 66
  - in regular expressions, 82–83
  - in XPath, 140

{"} (curly braces)
  - in annotations, 929
  - in glob patterns, 66
  - in message formatting, 336, 338–340
  - in regular expressions, 84

@ (at)
  - in annotations, 919
  - in URIs, 211
  - in XPath, 140

$ (dollar sign)
  - in native method names, 991
  - in regular expressions, 82–83, 86, 92

${{}} in policy files, 833

* (asterisk)
  - in annotations, 938
  - in DTDs, 116
  - in glob patterns, 66
  - in masks, 474
  - in permissions, 832–833
  - in policy files, 829
  - in regular expressions, 82, 84

\ (backslash)
  - in glob patterns, 66
  - in path names, 9, 57
  - in permissions (Windows), 832
  - in regular expressions, 82–84, 92
\|, in regular expressions, 17
\0, in regular expressions, 83
& (ampersand)
  in CDATA sections, 99
  in e-mail URIs, 714
  in entity references, 98
  in regular expressions, 83
parsing, 116
&amp;#, &amp;#x, in character references, 98
# (number sign)
  in masks, 473–474
  in message formatting, 338
  in URLs, 211
% (percent sign)
  in fortune program, 721
  in locales, 311
  in SQL, 243, 272
  in URLs, 224
%20, in URIs, 714
+ (plus sign)
  in DTDs, 116
  in regular expressions, 82, 84
  in URLs, 224
< (left angle bracket)
  in CDATA sections, 99
  in message formatting, 339
parsing, 116
<!-----> comments (HTML, XML), 99
<![CDATA[...]> sections, 99
<?...?> processing instructions (HTML, XML), 99
<> operator (SQL), 243
> (right angle bracket), in CDATA sections, 99
≤ operator (message formatting), 339
= operator (SQL), 243
== operator, 51
| (vertical bar)
  as delimiter in text files, 16
  in DTDs, 116–117
  in message formatting, 338
  in regular expressions, 82, 84
1931 CIE XYZ (color specification), 621
2D graphics
  context settings, 552
  printing, 637–647
XX20.Double classes, 554
XX20.Float classes, 554
3D rectangles, 553

A
A, in masks, 474
\a, \A, in regular expressions, 83–84
abort method (LoginModule), 838
absolute method (ResultSet), 276, 280
AbstractCellEditor class, 411–413
  isCellEditable method, 412
AbstractFormatter class
  getDocumentFilter method, 470, 483
  stringToValue, valueToString methods, 475, 483
AbstractListModel class, 371
AbstractProcessor class, 938
AbstractSpinnerModel class
  fireStateChanged method, 487, 494
  getNextValue, getPreviousValue methods, 487–488, 494
  get/setValue methods, 487, 494
AbstractTableModel class, 386
  isCellEditable method, 410
Action listeners
  annotating, 920–926
  constructing with the EventHandler, 782
  for text fields, 465
  installing, 921
  naming, 743
ActionEvent class, 743
ActionListener interface, 743, 921
  actionPerformed method, 922
@ActionListenerFor annotation, 921, 933
ActionListenerInstaller class, 921
  action.properties file, 913
Activatable class
  constructor, 986
  exportObject method, 982, 986
  extending, 981
  register method, 983, 986
Activation mechanism, 980–987
  descriptors, 980–981
  groups, 982
ActivationDesc class, 987
ActivationGroup class, 987
ActivationGroupDesc class, 986
activation/rmid.policy, 986
ActivationSystem interface, 987
activation/WarehouseActivator.java, 984
activation/WarehouseImpl.java, 985
add method
  of Area, 570–572
  of AttributeSet, 667
  of DefaultMutableTreeNode, 424, 431
  of SystemTray, 720, 723
addActionListener method
  injecting into frame constructor, 922
  of TrayIcon, 724
  using annotations instead of, 921
addAttribute method
  (AttributesImpl), 183
addBatch method
  (Statement), 298–300
addCellEditorListener method
  (CellEditor), 420
addChangeListener method
  (JTabbedPane), 520, 524
addColumn method
  (JTable), 398, 405
addDocumentListener method
  (Document), 464, 467
addElement method
  (DefaultListKeyModel), 377
addEventHandlers method, 913
addFlavorListener method
  (Clipboard), 680
addHyperlinkListener method
  (JEditorPane), 500
addListSelectionListener method
  (JList), 369
addPropertyChangeListener method
  implementing, 745
  of Component class, 746, 753
  of Customizer, 772
  of PropertyChangeSupport class, 751
addReceiver method
  (JavaMail), 231
add/remove/get naming pattern, 743, 745
addTab method
  (JTabbedPane), 518, 523
addTreeModelListener method
  (TreeModel), 454, 462
addTreeSelectionListener method
  (JTree), 445
addVetoableChangeListener method
  implementing, 747
  of JComponent, 543, 753
  of VetoableChangeSupport, 752
AES (Advanced Encryption Standard), 881
  generating keys, 882–887
aes/AESTest.java, 884
aes/Util.java, 885
Affine transformations, 586
  on pixels, 627
AffineTransform class, 586–589
  constructor, 587
  getXXXInstance methods, 586, 588
  setToXXX methods, 587, 589
AffineTransformOp class, 627
  constructor, 635
afterLast method
  (ResultSet), 276, 280
Agent code, 950
aliases method
  (Charset), 21, 24
Aliases
  for ISO 8859–1, 20
  for namespaces, 122, 148
distribute method
  (ByteBuffer), 76, 78
AllPermission class, 831
Alpha channel, 592
Alpha composites, 601
  precise computation of, 603
AlphaComposite class, 594
  getinstance method, 595, 601
AlreadyBoundException, 965
&amp; entity reference, 98
Anchor rectangle, 582
andFilter method
  (RowFilter), 397, 407
AnnotatedElementException
  getAnnotation method, 921, 926
  getAnnotations, getDeclaredAnnotations, isAnnotationPresent methods, 926
Annotation interface
  annotationType, equals, hashCode, toString methods, 930
  extending, 928
Annotation interfaces, 920, 926, 928
  methods of, 928
  predefined, 930–935
  proxies for, 928
Annotation processors, 937–940
  at bytecode level, 922, 943
  at runtime, 922
  compiling, 939
Annotations, 918–926
  applicability of, 933
  documented, 934–935
  elements of, 920, 928–929
  for compilation, 932
  for local variables, 930
  for managing resources, 932
  for other annotations, 931
  for packages, 930
  inherited, 934
  marker, 927
  meta, 920, 933–935
  multiple, for the same item, 930
  no circular dependencies in, 929
  processing tools for, 919
  retaining, 933
Annotations (cont.)
single value, 922
source-level, 935–45
standard, 930–35
syntax of, 926–30
transient, 934
vs. Javadoc comments, 918
Antialiasing, 601–4
ANY rule (DTDs), 116
Apache, 93, 100
Apache Derby database, 245
connections to, 249
default user name/password for, 249
driver for, 246
populating, 258–62
registering driver class for, 248
starting, 247–48
Apache Tomcat server, 302
'entity reference (HTML, XML), 98
appendChild method (Node), 160, 163
Applets
accessing browser’s cache, 214
class loaders for, 508
code base for, 524
extecuting from secure intranet, 585
not exiting JVM, 521
popping up dialog boxes, 214
running in browser, 876
security mechanisms in, 802–892
signed, 804, 873–880
using JDBC in, 302
warning windows in, 837
appletviewer program, 527
Application servers, 607, 632
Applications
communicating with native code, 689–707
configuring, 241–42
creating in Visual Basic, 723
data transfers between, 774
deploying with RMI, 565–68
desktop, 713–14
temporal, 41
enterprise, 301–303
loading time of, 708
localizing, 61
monitoring, 950
signing, 503, 573–80
splash screen for, 708
startup plugins for, 708
updates of, 715
with internal frames, 524
applyPattern method (MessageFormat), 337
apply-templates element (XSLT), 174
apt program, 937
Arc2D class, 553, 556, 557
Arc2D.Double class, 569
ArcMaker class, 563
Arcs
bounding rectangle of, 554–556
closure types of, 556
computing angles of, 556
Area class
add method, 570–72
exclusiveOr method, 571–72
intersect method, 571–72
subtract method, 571–72
Areas, 570, 77
ARGB color model, 594, 522
ARRAY data type (SQL), 301–03
array method (ByteBuffer, CharBuffer), 24
Array types, 1019, 1022
ArrayIndexOutOfBoundsException, 1022
Arrays
accessing elements from, 1024
from native code, 1019, 1022
bounds checking on, 807
creating in native methods, 1022
getting from a database, 300
in annotation elements, 929
of primitive types, 1022
type code for, 44, 1011
using JavaBeans persistence for, 784
ArrayStoreException, 1024
asCharBuffer method (ByteBuffer), 76
ASCII encoding
converting native encodings to, 331
in property files, 334
in regular expressions, 83
ASP (Active Server Pages), 222
ATTLIST rule (DTDs), 117
attribute element (XML Schema), 124
Attribute interface, 654
getCategory method, 564, 657
getName method, 567
implementing, 656
Attribute sets, 666
- adding/retrieving attributes, 667

Attributes
- enumerating, 104
- for enumerated types, 118
- for printing, 664–672
- in HTML, 98
- in XML Schema, 124
- legal, 117
- names of, 96
- namespace of, 148
- values of, 96
  - accessing in XPath, 140
  - copying with XSLT, 175
  - default (DTDs), 118
  - normalizing, 118
- vs. elements, 97, 118

Attributes interface
- getLength, getLocalName, getQName, getURI, getValue methods, 155

AttributeSet interface, 665
- add, get methods, 667, 672
- remove, toArray methods, 672

AttributesImpl class
- addAttribute, clear methods, 183

AudioPermission class, 831

auth/AuthTest.java, 846
auth/AuthTest.policy, 847

Authentication, 842–858
- problems of, 868–869
- role-based, 849
- separating from business logic, 843
- through a trusted intermediary, 869–870

auth/jaas.config, 847

AuthPermission class, 832

auth/SysPropAction.java, 846

Autoboxing, 382

AutoCloseable interface, 7
- close method, 7

Autocommit mode, 297
- turning off, 299

Autoflushing, 14

Autogenerated keys, 273–274

Automatic registration, 248

Auxiliary files, automatic generation of, 919
- available method (InputStream), 2, 4, 510
- availableCharsets method (Charset), 21, 24
- avoid keyword, 836

AWT (Abstract Window Toolkit), 549–724
- using JavaBeans persistence for, 784

AWTPermission class, 830, 837

B
\b, \B, in regular expressions, 83–84

B (byte), type code, 44, 1011

Banding, 639

Banner class
- getPageCount, layoutPages methods, 648

Banners, printing, 647–649

Base64 class, 216

BASE64Encoder class, 216

BasicFileAttributes interface, 63–64
- creationTime, lastXxxTime methods, 64
- fileKey method, 64
- isRegularFile, isDirectory, isSymbolicLink methods, 64
- size method, 64

BasicPermission class, 829

BasicStroke class, 572–581
- constructor, 581

Batch updates, 298–300
- error handling in, 298

BCEL (Bytecode Engineering Library), 943–952

Bean Builder, 736

Bean info classes, 935–943
- creating, 758
- generated automatically, 919, 936
- naming, 754
- specifying properties in, 743

BeanDescriptor class, 772

BeanInfo interface
- getBeanDescriptor method, 771
- getIcon method, 755–756
- getPropertyDescriptors method, 754–756, 762
- implementing, 754–758
- persistenceDelegate attribute, 785

BeanInfoAnnotationProcessor class
- process method, 938

Beans, 726–727
- adding events to, 739
- in a builder environment, 732–739
- custom icons for, 755
- defined, 725
- descriptors of, 785
- implementing, 725
- importing into NetBeans, 732
- info classes for, 741
- no-argument constructors for, 728
- packaging in JAR files, 731–733
- rich behavior of, 728
- usability of, 728
- writing, 727–731
Beans class, 740
beforeFirst method (ResultSet), 276, 280
Bevel join, 572–573
Bézier curves, 559
Big5 encoding, 23
BIG_ENDIAN constant (ByteOrder), 76
Big-endian order, 25
Bilinear interpolation, 627
Binary data
converting to Unicode code units, 13, 20
reading, 26
vs. text, 13
writing, 25
bind method
of Context, 964
of Naming, 965
Bindings interface, 896
get, put methods, 897
BitSet interface, 787
Blob interface, 269
getBytes method, 269–270
get/setBinaryStream methods, 269–270
length method, 270
BLOB data type (SQL), 245, 301
BLOBs (binary large objects), 269
creating empty, 271
placing in database, 270
retrieving, 269
Blocking
by I/O methods, 2
by network connections, 186, 190, 202–209
Blur filter, 629
BMP image format, 608
Book class, 647
append, getPrintable methods, 659
book/Banner.java, 653
book/BookTestFrame.java, 650
book/PrintPreviewCanvas.java, 657
book/PrintPreviewDialog.java, 655
BOOLEAN data type (SQL), 245, 301
boolean type
for property values, 742
printing, 14
type code for, 44, 1011
vs. C types, 997
writing in binary format, 25
Bootstrap class loader, 805
Bootstrap registry service, 961
Borders, using JavaBeans persistence for, 784
Bound properties, 745–746
Boundary matchers, 83
Bounding rectangle, 554
Bray, Tim, 96
Breadth-first enumeration, 440–441
breadthFirstEnumeration method
(_DefaultMutableTreeNode), 440–441, 444
browse method (Desktop), 719
Browsers, 495, 820
accessing password-protected pages in, 214
cache of, 214
forms in, 222–230
launching from desktop applications, 714
response page in, 223
running applets in, 875
Buffer class, 77–79
capacity method, 79
clear method, 78
flip method, 78
hasRemaining method, 74
limit method, 75
mark method, 78
position method, 79
remaining method, 78–79
reset method, 78
rewind method, 78
BufferedImage class, 582, 619
constructor, 619, 625
ggetColorModel method, 621, 625
gGetRaster method, 620, 625
BufferedImageOp, 619
filter method, 627, 635
implementing, 627
BufferedInputStream class, 12
BufferedOutputStream class, 12
BufferedReader class, 16
readLine method, 16
Buffers, 77–79
capacity of, 77
flushing, 3, 14
in-memory, 7
limits of, 77
marks in, 77
positions in, 70, 77
traversing all bytes in, 70
vs. random access, 69
BufferUnderflowException, 75
Builder environment, 731–739
adding new beans in, 732
and source code updates, 736
composing beans in, 733–739
icons in palette in, 755
buildSource method, 914
Business logic, 238
Butt cap, 572–573
Button listeners, for a dialog box, 469
ButtonType interface, 901, 921
buttons1/ButtonFrame.java, 906
buttons2/action.properties, 915
buttons2/ButtonFrame.java, 915
buttons3/ButtonFrame.java, 924
byte type
  turning negative values into unsigned
    integer, 475
type code for, 44, 1011
  vs. C types, 997
BYTE_ARRAY data source (print services), 661
ByteArrayOutputStream class, 54, 909
ByteBuffer class, 69, 77–79
  allocate method, 76, 78
  array method, 24
  asCharBuffer method, 76
  get method, 70, 75
  getXXX methods, 70, 76
  order method, 70, 76
  put method, 75
  putXXX methods, 70, 76
  wrap method, 24, 76, 78
bytecodeAnnotations/EntryLogger.java, 945
bytecodeAnnotations/EntryLoggingAgent.java, 951
Bytecodes
  engineering, 943–952
    at load time, 949–952
    with hex editor, 819
    verification of, 815–821
ByteLookupTable class, 628
  constructor, 635
ByteOrder class
  BIG_ENDIAN, LITTLE_ENDIAN constants, 76
Bytes, reading/writing, 2–4
C (char), type code, 44, 1011
C programming language
  array types in, 1019–1023
  bootstrap class loader in, 805
  calling Java methods from, 1012–1019
  database access in, 236
  embedding JVM into, 1028
FILE* type in, 4
  header file generating, 992
  invalid pointers in, 990
  strings in, 999
  types, vs. Java types, 997
    using with CORBA/SOAP, 956
C# programming language
  naming bean properties in, 741–742
    using with CORBA/SOAP, 956
C++ programming language
  accessing JNI functions in, 1000
  array types in, 1020
  embedding JVM into, 1028
  exceptions in, 1024
  for native methods, 990
  implementing native code, 993
  invalid pointers in, 990
  pointers to dummy classes in, 1017
  using with CORBA/SOAP, 956
CA (certificate authority), 871
Cache, 214
Cached row sets, 282–287
CachedRowSet interface, 282–286
  acceptChanges method, 284–285
  execute method, 283, 285
  get/setPageSize method, 283, 285
  get/setTableName method, 284–285
  nextPage method, 283, 285
  populate method, 283, 285
  previousPage method, 285
CachedRowSetImpl class, 282
CalendarBean class, 728, 730
call escape (SQL), 272
call method (CompilationTask), 909, 912
Call stack, during permission checking, 825
Callable interface, 909
Callback interface, 850
CallbackHandler interface
  handle method, 857
CallNonvirtualXXXMethod functions (C), 1018–1019
CallStaticXXXMethod functions (C), 1016–1017, 1019
CallXXXMethod functions (C), 1013, 1018
cancelCellEditing method (CellEditor), 413–414, 420
cancelRowUpdates method (ResultSet), 278, 281
canImport method (TransferHandler), 699, 705
canInsertImage method (ImageWriter), 612, 619
CANON_EQ flag (Pattern), 86
capacity method (Buffer), 79
CA.pl file, 872
Carriage return character, displaying, 104
Cascading windows, 527–528
CASE_INSENSITIVE flag (Pattern), 86
Catalogs (in databases), 295
CDATA attribute type (DTDs), 117–118
CDATA sections, 99
Cell editors, 410–411
custom, 411–420
Cell renderers
for lists, 377–381
for tables, 391, 408
for trees, 442–445
CellEditor interface
add/removeCellEditorListener methods, 420
cancelCellEditing method, 413–414, 420
ggetCellEditorValue method, 411, 413–414, 420
isCellEditable method, 420
shouldSelectCell method, 413, 420
stopCellEditing method, 413–414, 420
Cells (Swing)
editing, 410–411
rendering, 408–420
selecting, 394–395, 408
Certificates, 842, 865–868
and Java Plug-in, 873
for software developers, 878–880
publishing fingerprints of, 867
root, 874
set of, 822
signing, 870–873
CertificateSigner class, 871
CGI (Common Gateway Interface), 222
Chain of trust, 869
ChangeListener interface, 520
changeUpdate method (DocumentListener), 464, 467
Channels, 202
for files, 69
Channels class
newInputStream method, 209
newOutputStream method, 203, 209
char type
type code for, 44, 1011
vs. C types, 997
CHARACTER (CHAR) data type (SQL), 245, 301
Character classes, 82
Character encodings, 13, 21
basic, 22
converting to ASCII, 341
extended, 23
no connection between locales and, 340
of operating system, 340
required, 22
supported, 340
Character references, 98
Character sets, 20–25
CharacterData interface
data method, 103, 112
characters method (ContentHandler), 150, 155
Characters
antialiasing, 602
differences between, 329
escaping, 17, 82, 84
in regular expressions, 85
normalizing, 330
outlines of, 590
printing, 14
turning to upper/lower case, 471
writing in binary format, 25
CHAR_ARRAY data source (print services), 661
charAt method (ByteBuffer), 25
CharBuffer class, 7–8, 77
array method, 24
charAt method, 25
get, put methods, 76
toString method, 24–25
CharSequence interface, 8, 86
charAt, length methods, 8
subSequence, toString methods, 9
Charset class, 20–25
aliases method, 21, 24
availableCharsets method, 21, 24
character encoding names in, 22
decode, encode methods, 24
forName method, 21, 24
chart2/ChartBeanCustomizer.java, 774
ChartBean class
customizer for, 771
using NetBeans color property editor in, 773
ChartBeanInfo class, 759, 936
chart/ChartBeanInfo.java, 759
chart/InverseEditor.java, 767
chart/InverseEditorPanel.java, 768
chart/TitlePositionEditor.java, 765
Checkboxes
editing, 411
in table cells, 408
checked attribute (HTML, XML), 96
Checked exceptions, 844
checkError method (PrintWriter), 14, 16
checkExit method (SecurityManager), 821, 823–824
checkLogin method (SimpleLoginModule), 849
checkPermission method (SecurityManager), 823–825, 835–836
checkRead method (SecurityManager), 825
Child elements (XML), 97
  getting node list of, 104
  namespace of, 147
Child nodes (in trees), 103, 421
  adding, 424
  connecting lines for, 426–427
children method (TreeNode), 439
choice element (XML Schema), 124
choice keyword (message formatting), 338
1931 CIE XYZ color specification, 621
Cipher class, 881–882
  doFinal method, 882, 884, 887–888
  getBlocksize method, 886
  get Instance method, 881, 886
  getOutputsize method, 886
  init method, 886
  update method, 881, 884, 886, 888
  Xxx_MODE modes, 881
CipherInputStream class
  constructor, 888
  read method, 888
CipherOutputStream class, 887
  constructor, 888
  flush, write methods, 888
Ciphers
  generating keys, 882–887
  public keys in, 888–892
  streams for, 887–888
  symmetric, 881–882
Circular dependencies, in annotations, 929
Class class
  getClassLoader method, 805, 815
  getFields method, 456
  getProtectionDomain method, 825
  implementing AnnotatedElement, 921
  using JavaBeans persistence for, 784
.class file extension, 804
Class files, 804
  corrupted, 817, 820
  encrypted, 810–811
  format of, 943
  loading, 804
  location of, 908
  portability of, 340
  producing on disk, 909
  storing in databases, 908
  transformers for, 950
  verifying, 815–821
Class loaders, 804–815, 914
  as namespaces, 808
  bootstrap, 805
  context, 806–808
  creating, 821
  extension, 805
  hierarchy of, 806–808
  separate for each web page, 808
  specifying, 806
  writing, 808–815
Class path parameter (MIME types), 678
Class path hell, 805
Classes
  adding validation to, 48
  annotating, 920, 931
  compatibility with earlier versions of, 52–54
  descriptions of, 42
  externalizable, 44
  inheritance trees of, 441
  loading at runtime, 974
  locales supported by, 309
  nonserializable, 48
  persistence delegates for, 784–786
  protection domains of, 824
  resolving, 804
  separate for each web page, 808
  serializable, 37, 972
  undocumented, 216
ClassLoader class, 805
  defineClass method, 810, 815
  extending, 808
  findClass method, 809, 815
  getParent, getSystemClassLoader methods, 815
  loadClass method, 807, 809
ClassLoader inversion, 806
classpath/Caesar.java, 814
classLoader/ClassLoaderTest.java, 811
CLASSPATH environment variable, 805, 967
clear method
  of AttributesImpl, 183
  of Buffer, 78
CLEAR composition rule, 594
clearParameters method (PreparedStatement), 269
Clients
  accessing remote objects, 962
  configuring securely, 876
  connecting to servers, 185–193
  enumerating registered RMI objects, 961
  in distributed programming, 953–956
  installing proxies on, 955
  loading additional classes at runtime, 974
  multiple, serving, 197–201
  stubs on, 957–958, 962
Client/server applications, 238
clip method (Graphics2D), 551, 589–592, 639
Clipboard, 672–689
  accessing, 821
  capabilities of, 673
  transferring
    images, 680
    object references, 689
    objects, 685–689
    text, 674–678
Clipboard class, 674
  addFlavorListener method, 680
  constructor, 689
  getAvailableDataFlavors method, 680
  getData method, 678
  get/setContents methods, 674, 677
  isDataFlavorAvailable method, 678
ClipboardOwner interface
  implementing, 674
  lostOwnership method, 674, 678
Clipping region
  printing, 639
  setting, 551
Clob interface, 269
  get/setCharacterStream methods, 271
  getSubString method, 271
  length method, 271
CLOB data type (SQL), 245, 301
CLOBs (character large objects), 269
  creating empty, 271
  placing in database, 270
  retrieving, 270
clone method (Object), 37, 54
  and remote objects, 980
CloneNotSupportedException, 980
Cloning, 54–57
close method
  of AutoCloseable, 7
  of Closeable, 6, 8
  of Connection, 253, 255, 303
  of FileLock, 81
  of Flushable, 6
  of InputStream, 3–4
  of OutputStream, 4
  of ProgressMonitor, 505, 514
  of ResultSet, 255
  of ServerSocket, 197
  of SplashScreen, 713
  of Statement, 254–255
  of XMLStreamWriter, 172
Closeable interface, 6
close method, 6, 8
flush method, 6
Closed nonleaf icons, 428–430, 442
closed property (JInternalFrame), 746
closeEntry method
  ofZipInputStream, 33–34
  ofZipOutputStream, 34–35
closeOnCompletion method (Statement), 254
closePath method (Shape classes), 559, 570
Closure types, 556
Code base, 822, 828
Code generation
  annotations for, 918–926, 932
  in building tool, 764
Code Page 437, 341
Code signing, 804, 873–880
Code source, 822
  mapping to permission sets, 822
codeBase keyword, 828
Codebase URLs, 968
Codebreakers, The (Kahn), 810
CodeSource class
  getCertificates method, 826
  getLocation method, 826
Collation, 328–335
  strength of, 329
collation/CollationTest.java, 332
CollationKey class
  compareTo method, 335
Collator class, 329
  compare, equals method, 335
  getAvailableLocales method, 335
  getCollationKey method, 330, 335
  getInstance method, 335
  get/setDecomposition methods, 335
  get/setStrength methods, 335
Collections class
  sort method, 329
Collections, using JavaBeans persistence for, 784
Color class, 581
  constructor, 626
  getRGB method, 626
  translating values into pixel data, 623
  using JavaBeans persistence for, 784
Color chooser, 411, 685–689
Color space conversions, 629
ColorConvertOp class, 627, 629
ColorModel class, 623
  get/setDecomposition methods, 335
  get/setStrength methods, 335

Colors
components of, 592
composing, 592–593
dithering, 602
interpolating, 581
negating, 628
solid, 550

Columns (in databases)
accessing by number, in result set, 253
names of, 239
number of, 287

Columns (Swing)
accessing, 392
adding, 399
detached, 383
hiding, 398–407
names of, 388
rendering, 390–391
resizing, 383–384, 392–393
selecting, 394

Combo boxes, 364
editing, 411
for property settings, 763
traversal order in, 486

Comma-separated files, 226

Comments (XML), 99
commit method
of Connection, 297–299
of LoginModule, 858

Commit or revert behavior, 469
commitEdit method (JFormattedTextField), 469, 482
Comparable interface, 396
Comparator interface, 329
Comparators, 396
compare method (Collator), 335
compareTo method
of CollationKey, 335
of Comparable, 396
of String, 328
Compatibility characters, 330
Compilable interface, 901
compile method, 901

Compilation tasks, 907–913
CompilationTask interface, 907–909
call method, 909, 912
compile method (Pattern), 86, 91
CompiledScript class
eval method, 901

Compiler
annotations for, 932
invoking, 907
just-in-time, 1029
producing class files on disk, 909

Compiler API, 906–918

compiler/ByteArrayJavaClass.java, 910
compiler/CompilerTest.java, 916
compiler/MapClassLoader.java, 918
compiler/StringBuilderJavaSource.java, 910

Complex types (XML Schema), 122
complexType element (XML Schema), 123

Component class
add/removePropertyChangeListener methods, 746, 753
extending, 740, 746, 772
firePropertyChange method, 753
getPropertyChangeListener method, 746

Composite interface, 594

composite/CompositeComponent.java, 598
composite/CompositeTestFrame.java, 597
composite/Rule.java, 599

Composition rules, 550, 592–601
setting, 551

Computer Graphics: Principles and Practice
(Foley), 559, 594, 621

com.sun.rowset package, 282
com.sun.security.auth.module package, 843

Confidential information, transferring, 880
Configuration files, 79
connect method
of Socket, 191
of URLConnection, 212, 214, 221

Connection interface
close method, 253, 255, 303
commit method, 297–299
createBlob, createClob methods, 270–271
createStatement method, 252–253, 274, 279, 297
getAutoCommit method, 299
getMetaData method, 286, 295
getWarnings method, 258
prepareStatement method, 263, 269, 274–275, 279
releaseSavepoint method, 298–299
rollback method, 297–299
setSavepoint method, 299

Connection pool, 303

Connections
closing, 255
using row sets after, 282
starting new threads, 198

Constrained properties, 746–754

Constructive area geometry operations, 570
Constructor class, 921
@ConstructorProperties annotation, 786–787
Constructors
  invoking from native code, 1017
  no-argument, 728
containsAll method (Set), 837
Content types, 213
ContentHandler interface, 150–151
  characters method, 150, 155
  start/endDocument methods, 154
  start/endElement methods, 150–155
Context interface
  bind, unbind, rebind methods, 964
  list method, 964
  lookup method, 964
Context class loader, 806–808
CONTIGUOUS_TREE_SELECTION mode
  (TreeSelectionModel), 446
Control points
  dragging, 561
  of curves, 556
  of shapes, 560
Controls (Visual Basic), 726
convertXxxIndexToModel methods (JTable), 395, 405
Convolution operation, 629
ConvolveOp class, 627, 629–630
  constructor, 636
Coordinate system
  custom, 551
  translating, 640
Coordinate transformations, 583–589
Copies class, 665–667
  getValue method, 667
CopiesSupported class, 665
copy method (Files), 61–62
CORBA (Common Object Request Broker
  Architecture), 956
Java support of, 805
Core Swing (Topley), 382, 421, 434, 463
count function (XPath), 141
Count of Monte Cristo, The (Dumas), 510
Country codes, 307–308
CRC32 class, 71
CRC32 checksum, 35–36, 69, 71
CREATE TABLE statement (SQL), 244
  executing, 252, 254, 269
  in batch updates, 298
createBindings method (ScriptEngine), 897
createBlob, createClob methods (Connection), 270–271
createFile, createDirectory, createDirectories methods (Files), 62–63
createGraphics method (SplashScreen), 713
createImageXxxStream methods (ImageIO), 610, 617
createPrintJob method (PrintService), 660, 662
createStatement method (Connection), 252–253, 274, 279, 297
createTempXxx methods (Files), 62–63
createTransferable method (TransferHandler), 696, 698
createXMLStreamReader method (XMLInputFactory), 158
createXxxWriter method (XMLOutputFactory), 164, 171
createXxx methods (Document), 160, 163
createXxxxRowSet methods (RowSetFactory), 282, 286
creationTime method (BasicFileAttributes), 64
Credit card numbers, transferring, 880
crypt method, 883
Cryptography and Network Security (Stallings), 43, 860
Cubic curves, 556, 559
CubicCurve2D class, 553, 556
CubicCurve2D.Double class, 569
Currencies
  available, 319
  formatting, 311–319, 472
  identifiers for, 318
Currency class, 318–319
  getAvailableCurrencies method, 319
  getCurrencyCode method, 319
  getDefaultFractionDigits method, 319
  getInstance method, 318–319
  getSymbol method, 319
  toString method, 319
Cursor class, using JavaBeans persistence for, 784
Cursor, drop shapes of, 690–691
curveTo method (shape classes), 559, 570
Custom editors, 411, 765
Custom formatters, 474–485
Customizer interface
  add/removePropertyChangeListener methods, 772
  implementing, 772
  setObject method, 772–773, 779
Customizers, 770–779
  editing property values in, 773
  naming pattern for, 771
  opening in NetBeans, 772
tabbed panes for, 772
writing, 772–779
Cut and paste, 672–689
Cygwin programming environment, 994
compiling invocation API, 1033
OpenSSL in, 872
D
\d, \D, in regular expressions, 83
D (double), type code, 44, 1011
damageReporter/DamageReporterFrame.java, 793
damageReporter/DamageReport.java, 798
Dashed lines, dash pattern, dash phase, 574
Data flavors, 678–680
human-readable names of, 678
MIME types names in, 678
representation classes in, 678
Data sources, 302
Data transfer, 672–689
between Java and native code, 673
classes and interfaces for, 674
in Swing, 691–707
Data types
codes for, 44, 1011
mangling names of, 1010
print services for, 659–661
DatabaseMetaData interface, 286–296
getJDBCXxxVersion methods, 296
getMaxConnection method, 296
getMaxStatements method, 255, 296
getSQLStateType method, 256
getTables method, 286, 295
supportsBatchUpdates method, 298, 300
supportsResultSetXXX methods, 276, 281
database.properties file, 259, 302
Database, 235–303
accessing, in C language, 236
autonumbering keys in, 273
avoiding duplication of data in, 241
batch updates for, 298–300
caching prepared statements, 264
changing data with SQL, 244
connections to, 246, 259
closing, 255, 260
in web and enterprise applications, 301–303
pooling, 303
drivers for, 236–237
integrity of, 297
LOBs in, 269–271
metadata for, 286–296
modifying, 283
native storage for XML in, 301
numbering columns in, 253
outer joins in, 272
populating, 258–262
saving objects to, 935
scalar functions in, 272
schemas for, 295
setting up parameters in, 283
starting, 247–248
stored procedures in, 272
storing
class files in, 908
logins in, 849
structure of, 239, 286
synchronization of, 284
tools for, 287
truncated data from, 257
URLs of, 246
DataFlavor class, 674, 678–680
constructor, 679
getHumanPresentableName method, 680
getMimeType method, 679
getRepresentationClass method, 680
imageFlavor constant, 680–681
isMimeTypeEqual method, 679
javaFileListFlavor constant, 700
DataInput interface, 26
readFully method, 27
readUTF method, 26–27
readXXX methods, 26–27, 37
skipBytes method, 27
DataInputStream class, 5, 10
Data class
readObject method, 49
writeObject method, 49
DataSource interface, 302
DataSource class, 5
DataSource class
getTransferSize method, 258
DATE data type (SQL), 245, 271, 301
Date picker, 487
DateEditor class, 494
dateFilter method (RowFilter), 397, 407
DateFormat class, 319–328
  constants, 320
  format method, 320, 326
  getAvailableLocales method, 319, 326
  get/setCalendar methods, 327
  get/setNumberFormat methods, 327
  get/setTimeZone methods, 327
  get/setXxxInstance methods, 319, 326, 472
  is/setLenient methods, 326, 473
  parse method, 320, 326
  supported locales, 309
dateFormat/DateFormatTest.java, 321
dateFormat/EnumCombo.java, 325
Dates
  displaying in a calendar, 728
  editing, 472
  filtering, 397
  formatting, 306, 319–328
  lenient, 320, 473
  literals for, 271
DateTimeSyntax class, 668
Daylight saving time, 327
DDL (Data Definition Language), 254, 269
Debugging
  in JNI, 1029
  JDBC, 250
  of locales, 311
  policy files, 875
  with telnet, 185
Decapitalization, 742
DECIMAL (DEC) data type (SQL), 245, 301
Decimal separators, 306, 312, 468
decode method
  of Charset, 24
  of URLDecoder, 230
Decomposition, 329–335
Decryption key, 810
DECRYPT_MODE mode (Cipher), 881
default keyword, 926
DefaultCellEditor class, 435
  constructor, 419
  variations, 411
DefaultFormatter class
  extending, 475
  get/setOverwriteMode methods, 473, 483
  toString method, 473
DefaultHandler class, 151
DefaultListModel class, 376
  add/removeElement methods, 377
DefaultMutableTreeNode class, 423, 440–442
  add method, 424, 431
  breadthFirstEnumeration method, 440–441, 444
  constructor, 431
  depthFirstEnumeration method, 440–441, 444
  postOrderEnumeration method, 441, 444
  preOrderEnumeration method, 441, 444
  setAllowsChildren method, 428, 431
  setAsksAllowsChildren method, 430–431
defaultPage method (PrintJob), 646
DefaultPersistenceDelegate class, 786, 792
  constructor, 802
  initialize method, 787, 802
  instantiate method, 802
DefaultRowSorter class
  setComparator method, 396, 406
  setRowFilter method, 397, 406
  setSortable method, 395, 406
DefaultTableCellRenderer class
DefaultTableCellRenderer class, 408
DefaultTableModel class
  isCellEditable method, 410
DefaultTreeCellRenderer class, 442–445
  setXxxIcon methods, 445
DefaultTreeModel class, 432, 455
  automatic notification by, 434
  getPathToRoot method, 434
  insertNodeInto method, 433, 439
  isLeaf method, 428
  nodeChanged method, 433, 440
  nodesChanged method, 440
  reload method, 434, 440
  removeNodeFromParent method, 433, 439
defaultWriteObject method (ObjectOutputStream), 49
defineClass method (ClassLoader), 810, 815
Delayed formatting, 674
delete method (Files), 61–62
DELETE statement (SQL), 244
  executing, 252, 254, 269
  in batch updates, 298
  vs. methods of ResultSet, 278
DeleteGlobalRef function (C), 1007
deleteIfExists method (Files), 62
deleteRow method (ResultSet), 278, 281
Delimiters, in text files, 16
Deprecated interface, 931
@Deprecated annotation, 932
Depth-first enumeration, 440–441
depthFirstEnumeration method
   (DefaultMutableTreeNode), 440–441, 444
DER (distinguished encoding rules), 872
derbyclient.jar file, 246
DES (Data Encryption Standard), 881
Descriptor class
   setPropertyEditorClass method, 937
Desktop class, 707, 713–719
   browse method, 719
   edit method, 714, 719
   getDesktop method, 714, 719
   isDesktopSupported method, 714, 719
   isSupported method, 714, 719
   mail method, 719
   open method, 714, 719
   print method, 714, 719
Desktop applications, launching, 713–719
Desktop panes, 524–527
   cascading/tiling, 527–531
desktopApp/DesktopAppFrame.java, 714
DesktopManager class, 534
DestroyJavaVM function (C), 1029, 1034
Device coordinates, 584
Diagnostic interface
   getKind method, 912
   getMessage method, 912
   getSource method, 912
   getXXXNumber methods, 912
DiagnosticCollector class, 908
   constructor, 912
   getDiagnostics method, 912
DiagnosticListener interface, 908
DialogCallbackHandler class, 850
Dialogs
   cascading/tiling, 527–531
   in internal frames, 533
   validating input fields before closing, 469
digest method (MessageDigest), 861–862
Digital signatures, 858–873
   verifying, 865–868
Dimension class, using JavaBeans persistence for, 784
Document interface
   addDocumentListener method, 464, 467
   createXXX methods, 160, 163
   getDocumentElement method, 100, 111
   getLength method, 466
   getText method, 466
   implementing, 462
Document listeners, 464
DocumentBuilder class
   newDocument method, 159, 163, 179
   parse method, 111
   setEntityResolver method, 115, 121
   setErrorHandler method, 120–121
DocumentBuilderFactory class
   is/setIgnoringElementContentWhitespace methods, 119, 122
   is/setNamespaceAware methods, 124, 149, 152, 160
   is/setValidating method, 122
   newDocumentBuilder method, 100, 111, 160
   newInstance method, 100, 111
   setNamespaceAware method, 160
   setValidating method, 119
@Documented annotation, 934–935
Documented interface, 931
DocumentEvent interface
getDocument method, 467
DocumentFilter class, 471
insertString method, 470–471, 483–484
remove method, 484
replace method, 470–471, 484
DocumentListener interface
attaching to text field, 773
changeUpdate, insertUpdate, removeUpdate methods, 464, 467
doFinal method (Cipher), 882, 884, 887–888
DOM (Document Object Model), 99
DOM parser, 99–100, 150
DOM tree
accessing with XPath, 139–146
analyzing, 102–104
building, 159–172
without namespaces, 159–160
writing, 161
DOMResult class, 178
constructor, 183
DOMSource class, 177
constructor, 164
dom/TreeViewer.java, 105
DOTALL flag (Pattern), 86
DOUBLE data type (SQL), 245, 301
double type
printing, 14
type code for, 44, 1011
vs. C types, 997
writing in binary format, 25
DoubleClickEditor class, 759
DoubleBuffer class, 77
Double-clicking, on list components, 367
doubleValue method (Number), 312
Drop and drop, 689–707
file lists in, 700
moving vs. copying with, 690
supported in Swing, 691, 700–707
visual feedback for, 699
Drag sources, 690
configuring, 696–698
draw method (Graphics2D), 551–553, 571
Drawing
shapes, 550–553
simple, 549
drawXxx methods (Graphics), 553
DriverManager class, 248
getConnection method, 249, 251, 259, 303
setLogWriter method, 250
Drop actions, 690–691
Drop location, 700–701
DROP TABLE statement (SQL), 249
executing, 252, 254
in batch updates, 298
Drop targets, 690, 699–707
DropLocation class (of JList)
getIndex method, 701, 706
isInsert method, 701, 706
DropLocation class (of JTable)
getColumn, getRow methods, 707
isInsertColumn, isInsertRow methods, 707
DropLocation class (of JTable)
getIndex method, 707
DropLocation class (of JTree)
getChildIndex method, 707
getAddress method, 707
DropLocation class (of TransferHandler), 701
getDropPoint method, 706
DSA (Digital Signature Algorithm), 863–865
DST, DST_Xxx composition rules, 594
DTDHandler interface, 151
DTDs (Document Type Definitions), 113–122
element content in, 116–117
entities in, 119
external, 114
in XML documents, 96, 114
unambiguous, 117
URLs for, 114
Dynamic class loading, 974–979
Dynamic links, 1030
Dynamic web pages, 913–918

E
\e, \E, in regular expressions, 83–84
Echo server, accessing, 196
Eclipse, startup plugins for, 708
Edge detection, 629
edit method (Desktop), 714, 719
Editor pane, 463, 494–501
edit mode of, 496
loading pages in separate threads, 496
editorPane/EditorPaneFrame.java, 498
Editors, custom, 411
EJBs (Enterprise JavaBeans), 726
hot deployment of, 808
element element (XML Schema), 123
Element interface
getAttribute method, 104, 111, 126–127
getTagName method, 101, 111, 149
setAttribute, setAttributeNS methods, 160, 163
Elements (XML)
- child, 97
- accessing in XPath, 141
- constructor, 160
- counting in XPath, 141
- empty, 96
- getting node list of, 104
- legal attributes of, 117
- names of, 101, 149
- root, 97, 122
- trimming whitespace in, 103
- vs. attributes, 97, 118

Empty tags (XML), 96

Employer/Employee.c, 1009
Employee/Employee.java, 1008
Employee/EmployeeTest.java, 1008

Entries method (ZipFile), 36
Entry class, 398
- getIdentifier, getModel, getStringValue, getValue,
- getValueCount methods, 407
EntryLogger class, 950
EntryLoggingAgent.mf file, 950
enum keyword, 51
EnumCombo class, 324
enumeration element (XML Schema), 123
- attributes for, 118
Enumeration interface, 36
- hasMoreElements method, 1036, 1038
- nextElement method, 440, 1036, 1038
Enumerations
- of nodes, in a tree, 440–442
- typesafe, 50–52
- using JavaBeans persistence for, 784
EnumSyntax class, 668
EOFException, 1023
Epoch, 49
equals method
- and remote objects, 980
- of Annotation, 930
- of Collator, 335
- of Permission, 834
- of Set, 837
error method (ErrorHandler), 120–121

Error handlers
- in native code, 1023–1028
- installing, 120

Error messages
- control over, 909
- listening to, 908

ErrorHandler interface, 151
- error, fatalError, warning methods, 120–121
escape escape (SQL), 272
Escape hatch mechanism, 431
Escapes
- in regular expressions, 17, 82, 84
- in SQL, 271–272
Essential XML (Box et al.), 93, 174
Euro symbol, 20, 318
eval method
- of CompiledScript, 901
- of ScriptEngine, 895–897
evaluate method (XPath), 141, 146
Event handlers
- annotating, 920–926
- using JavaBeans persistence for, 784
Event listeners, 128, 919
naming, 743

Elements (XML)
- child, 97
- accessing in XPath, 141
- constructor, 160
- counting in XPath, 141
- empty, 96
- getting node list of, 104
- legal attributes of, 117
- names of, 101, 149
- root, 97, 122
- trimming whitespace in, 103
- vs. attributes, 97, 118

Empty tags (XML), 96

Employer/Employee.c, 1009
Employee/Employee.java, 1008
Employee/EmployeeTest.java, 1008

Entries method (ZipFile), 36
Entry class, 398
- getIdentifier, getModel, getStringValue, getValue,
- getValueCount methods, 407
EntryLogger class, 950
EntryLoggingAgent.mf file, 950
enum keyword, 51
EnumCombo class, 324
enumeration element (XML Schema), 123
- attributes for, 118
Enumeration interface, 36
- hasMoreElements method, 1036, 1038
- nextElement method, 440, 1036, 1038
Enumerations
- of nodes, in a tree, 440–442
- typesafe, 50–52
- using JavaBeans persistence for, 784
EnumSyntax class, 668
EOFException, 1023
Epoch, 49
equals method
- and remote objects, 980
- of Annotation, 930
- of Collator, 335
- of Permission, 834
- of Set, 837
error method (ErrorHandler), 120–121

Error handlers
- in native code, 1023–1028
- installing, 120

Error messages
- control over, 909
- listening to, 908

ErrorHandler interface, 151
- error, fatalError, warning methods, 120–121
escape escape (SQL), 272
Escape hatch mechanism, 431
Escapes
- in regular expressions, 17, 82, 84
- in SQL, 271–272
Essential XML (Box et al.), 93, 174
Euro symbol, 20, 318
eval method
- of CompiledScript, 901
- of ScriptEngine, 895–897
evaluate method (XPath), 141, 146
Event handlers
- annotating, 920–926
- using JavaBeans persistence for, 784
Event listeners, 128, 919
naming, 743
Event queues (AWT), 821
EventHandler class, 922
constructing action listeners with, 782
EventListenerList class, 455
EventObject class, 743
Events, 737, 742
adding to beans, 739
names of, 739–743
transitional, 366
Evins, Jim, 391
evn pointer (JNI), 1000
ExceptionCheck function (C), 1028
ExceptionClear function (C), 1024, 1028
ExceptionListener interface
exceptionThrown method, 801
ExceptionOccurred function (C), 1024–1025, 1028
Exceptions
checked, 844
from native code, 1023–1028
in C++, 1024
in SQL, 256–258
Exclusive lock, 80
exclusiveOr method (Area), 571–572
exec/ExecSQL.java, 260
execute method
of CachedRowSet, 283, 285
of RowSet, 283, 285
of Statement, 254, 259, 273–274
executeBatch method (Statement), 298, 300
executeQuery method
of PreparedStatement, 264, 269
of Statement, 252–253, 276
executeUpdate method
of PreparedStatement, 264, 269
of Statement, 252, 254, 274, 297
exists method (Files), 63–64
EXIT statement (SQL), 247
exit method (Runtime), 821–842
exportAsDrag method (TransferHandler), 697–698
exportDone method (TransferHandler), 698
exportObject method
of Activatable, 982, 986
of UnicastRemoteObject, 960
Expression class, 802
Extension class loader, 805
extensions property (FilePickerBean), 744
extern "C" keyword, 993
External entities, 119
Externalizable interface
readExternal, writeExternal methods, 50

F
\f, in regular expressions, 83
F (float), type code, 44, 1011
Factoring algorithms, 865
Factory methods, saving objects from, 787
fatalError method (ErrorHandler), 120–121
FeatureDescriptor class
get/setDisplayName methods, 757
get/setName methods, 757
get/setShortDescription methods, 757
is/setExpert methods, 757
is/setHidden methods, 757
Field class
getName, getType methods, 456
implementing AnnotatedElement, 921
using JavaBeans persistence for, 784
Fields
accessing
from another class, 821
from native code, 1005–1010
annotating, 920, 931
signatures of, 1006
transient, 48–49
File class, 64
toPath method, 60
File lists, 700
File permissions, 829
File pointers, 28
File systems, POSIX-compliant, 64
file: URL prefix, 210, 828
FileChannel class
lock, tryLock methods, 79–81
open, map methods, 69, 74
FileInputStream class, 9–12, 509, 824–825, 835
constructor, 11
getchannel method, 74
read method, 2
fileKey method (BasicFileAttributes), 64
FileLock class
close method, 81
isShared method, 80
fileName property
of FilePickerBean, 745
of ImageViewerBean, 744
FileNotFoundException, 227
FileOutputSteam class, 9–12
constructor, 11
getchannel method, 74
FilePermission class, 823, 829
FilePickerBean class, 748
extensions property, 744
filePicker property, 745
filePicker/FilePickerBean.java, 748
File interface, 938
FileReader class, 824–825
Files
  bean properties for, 744
  accessing, 821
  channels for, 69
  configuration, 79
  copying, 61
  counting lines in, 510
  creating, 62–63
  deleting, 61
  drag and drop of, 691–692
  encrypting/decrypting, 887–888
  filtering, 65–66
    by file suffixes, 609
  generated automatically, 919, 935
  handling, 909
  hierarchical structure of, 420
  in desktop applications, 714
  I/O modes of, 32
  iterating over, 64–67
  location of, 908
  locking, 79–81
    tail portion, 80
  memory-mapped, 68–81
  moving, 61
  names of, 341
  owners of, 63
  random-access, 28–33
    vs. buffered, 69
  reading, 60–61
    as a string, 60
    by one byte, 2–4
    numbers from, 10
    permissions for, 835
  source, 340–341
  textual, 340
  total number of bytes in, 29, 63
  with multiple images, 610–619
    reading, 610–611
    writing, 611
  working with, 57–68
  writing, 60–61
Files class, 57, 60–67
  copy method, 61–62
  createXxx methods, 62–63
  delete method, 61–62
  deleteIfExists method, 62
  exists method, 63–64
getBytes method, 60
getOwner method, 63
isXxx methods, 63–64
move method, 61–62
newDirectoryStream method, 65, 67
newXxxStream, newXxxmethods, 61
readAllXxx methods, 60–61
readAttributes method, 64
size method, 63–64
walkFileTree method, 65, 67
write method, 60–61
file.separator property, 833
FileSystem class
  getPath method, 68
FileSystems class
  newFileSystem method, 67–68
FileVisitor, 66–67
  postVisitDirectory method, 65
  preVisitDirectory method, 65
  visitFile, visitFileFailed methods, 65
FileWriter class, 13
  fill method (Graphics2D), 551–552, 571
  Filling, 550–551, 581
  fillXxx methods (Graphics), 553
  filter method (BufferedImageOp), 627, 635
  FilteredRowSet interface, 282
  FilterInputStream class, 10
  FilterOutputStream class, 10
Filters
  for images, 626–636
  for numbers, 397
  for streams, 509
  for table rows, 396–398
  for user input, 470–471
  glob patterns for, 65–66
  implementing, 397
find method (Matcher), 89, 92
findClass method (Classloader), 809, 815
findClass function (C), 1006, 1009, 1016
findColumn method (ResultSet), 255
findUserObject method, 442, 447
Fingerprints (SHA), 43, 858–873
  different for a class and its objects, 46
fireIndexedPropertyChange method
  (PropertyChangeSupport), 751
firePropertyChange method, 746
  of Component class, 753
  of JComponent class, 746
  of PropertyChangeSupport class, 751
fireStateChanged method (AbstractSpinnerModel), 487, 494
fireVetoableChange method
  of JComponent, 748, 753
  of VetoableChangeSupport, 747, 753
first method (ResultSet), 276, 280
#FIXED attribute default (DTDs), 118
Fixed-size patterns, 473
Fixed-size records, 29–30
FlavorListener interface
  flavorsChanged method, 679–680
flavormap.properties file, 673
flip method (Buffer), 78
FLOAT data type (SQL), 245, 301
float type
  printing, 14
  type code for, 44, 1011
  vs. C types, 997
  writing in binary format, 25
FloatBuffer class, 77
Floating-point numbers, 306, 311–319, 472
flush method
  of CipherOutputStream, 888
  of Closeable, 6
  of Flushable, 6, 8
  of OutputStream, 3–4
Flushable interface, 6–7
  close method, 6
  flush method, 6, 8
fn escape (SQL), 272
Focus listeners, 465
Folder icons, 428–430, 442
Font class, using JavaBeans persistence for, 784
Font render context, 590
Fonts
  antialiasing, 602–603
  displaying, 378
Forest, 421, 428–429
forLanguageTag method (Locale), 310
format method
  of DateFormat, 320, 326
  of Format, 337
  of MessageFormat, 337
  of NumberFormat, 313, 317
Formatters
  and locales, 307
  custom, 474–485
  installing, 470
  supported by JFormattedTextField, 472–474
Formatting
  copy and paste, 673
  delayed, 674
  of currencies, 472
  of dates, 306, 319–328
  of messages, 335–340
  of numbers, 306, 311–319, 472
Forms, processing, 222–230
forName method (Charset), 21, 24
fortune program, 720
ForwardingJavaFileManager class
  constructor, 913
  getFileForOutput method, 913
  getJavaFileForOutput method, 909, 914
fprintf function (C), 1025
fn printf function (C), 1013
Fractals, 622
Frames
  cascading/tiling, 527–531
  closing, 531
  dialogs in, 533
  dragging, 534
  grabbers of, 524
  icons for, 525
  internal, 524–534
  managing on desktop, 534
  moving selections between, 530
  positioning, 527
  resizable, 528
  selected, 527
  states of, 527–528
  visibility of, 526
  with two nested split panes, 515
FROM clause (SQL), 242
FTP (File Transfer Protocol), 214
ftp: URL prefix, 210, 214, 495
G
\G, in regular expressions, 84
Garbage collection
  and arrays, 1022
  and native methods, 1001
  of remote objects, 972
GeneralPath class, 553, 559
  constructor, 570
@Generated annotation, 932
Generated interface, 931
generateKey method (KeyGenerator), 882, 887
Gestures, 690
GET command, 224–225
get method
  of AttributeSet, 667, 672
  of Bindings, 897
  of ByteBuffer, 70, 75
of CharBuffer, 76
of MarshallledObject, 981, 986
of Paths, 58–59
of ScriptEngine, 897
of ScriptEngineManager, 897
getActions method (Permission), 834
getAddress method (Socket), 192–193
getAdvance method (TextLayout), 591
getAllByName method (Socket), 192–193
getAllFrames method (JDesktopPane), 527–530, 541
getAllowsChildren method (TreeNode), 430
getAllowUserInteraction method (URLConnection), 221
getAnnotation method (AnnotatedElement), 921, 926
getAnnotations, getDeclaredAnnotations methods
(AnnotatedElement), 926
GetArrayLength function (C), 1020, 1023
getAscent method (TextLayout), 592
getAsText method (PropertyEditor), 762–763, 769
overriding, 766
getAttribute method (Element), 104, 111, 126–127
getAttributes method
of DocPrintJob, 672
of Node, 104, 112
of PrintService, 672
getAttributeXxx methods (XMLStreamReader), 156,
159
getAuthority method (URI), 211
getAutoCommit method (Connection), 299
getAutoCreateRowSorter method (JTable), 384, 386
getAvailableCurrencies method (Currency), 319
getAvailableDataFlavors method (Clipboard), 680
getAvailableIDs method (TimeZone), 327
getAvailableLocales method
of Collator, 335
of DateFormat, 319, 326
of NumberFormat, 309, 313, 317
getBackground method (List), 378, 381
getBeanDescriptor method (BeanInfo), 771
getBeanInfo method (Introspector), 755–756
getBinaryStream method (Blob), 269–270
getBlob method (ResultSet), 269–270
getBlockSize method (Cipher), 886
getBounds method (SplashScreen), 713
getBundle method (ResourceBundle), 342–344, 346
getByIName method (Socket), 192–193
GetByteArrayElements function (C), 1038
getBytes method
of Blob, 269–270
of Clob, 271
of Files, 60
getCalendar method (Dateformat), 327
getCandidateLocales method
(ResourceBundle.Control), 343
getCategory method (Attribute), 666, 671
getCellEditor Value method (CellEditor), 411,
413–414, 420
getCellSelectionEnabled method (JTable), 405
getCertificates method (CodeSource), 826
getChannel method
of FileInputStream, 74
of FileOutputStream, 74
of RandomAccessFile, 74
getCharContent method (SimpleJavaFileObject),
909, 913
getchild method (TreeModel), 104, 454–456, 461
getchildAt method (TreeNode), 439
getchildCount method
of TreeModel, 454–456, 461
of TreeNode, 439
getchildIndex method (JTree.DropLocation), 707
getchildNodes method (Node), 101, 112
giclassLoader method (Class), 805, 815
getchildName method (NameClassPair), 965
getchip method (Graphics), 591, 639
getClO method (ResultSet), 269–270
getCodeSource method (ProtectionDomain), 826
getCollationKey method (Collator), 330, 335
getchromModell method (BufferedImage), 621, 625
getcholumn method
of JTable.DropLocation, 707
of TableModel, 405
getcholumnClass method (TableModel), 390, 404
getcholumnCount method
of ResultSetMetaData, 287, 296
of TableModel, 386, 388, 390
getcholumnModel method (JTable), 404
getcholumnName method (TableModel), 388, 390
getcholumnNumber method
of Diagnostic, 912
of SAXParseException, 121
getcholumnSelectionAllowed method (JTable), 405
getcholumnXxx methods (ResultSetMetaData), 287,
296
getchoMethod method (RowSet), 285
getchooMent method (TransferSupport), 706
getchooMentAt method (JTabbedPane), 523
getchooConcurrency method (ResultSet), 276–277, 280
getchooConnection method (DriverManager), 249, 251,
259, 303
getchooConnectionTimeout method (URLConnection), 221
getchooContent method (URLConnection), 213, 222
getContentPane method (JInternalFrame), 542
getcContents method (Clipboard), 677
gGetContentXxx methods (URLConnection), 213, 216, 222
gGetContext method
  of AccessController, 848
  of ScriptEngine, 898
gGetContextClassLoader method (Thread), 807, 815
gGetCountry method (Locale), 311
gGetCurrencyCode method (Currency), 319
gGetCurrencyInstance method (NumberFormat), 311, 317–318, 472
gGetCustomEditor method (PropertyEditor), 762, 767, 770
gGetData method
  of CharacterData, 103, 112
  of Clipboard, 678
gGetDataElements method
  of ColorModel, 626
  of Raster, 621–622, 625
gGetDataFlavors method (TransferSupport), 706
gGetDataSize method (DataTruncation), 258
gGetDate method (URLConnection), 213, 216, 222
gGetDateTimeInstance, getDateTimeTimeInstance methods
  (DateFormat), 319, 326, 472
gGetDecomposition method (Collator), 335
gGetDefault method
  of Locale, 309–310
  of TimeZone, 327
gGetDefaultEditor method (JTable), 419
gGetDefaultFractionDigits method (Currency), 319
gGetDefaultName method (NameCallback), 857
gGetDefaultRenderer method (JTable), 409, 419
gGetDescent method (TextLayout), 592
gGetDesktop method (Desktop), 714, 719
gGetDesktopPane method (JInternalFrame), 542
gGetDiagnostics method (DiagnosticCollector), 912
gGetDirectBufferAddress function (C), 1022
  GetDirectBufferCapacity function (C), 1022
getDirectBufferAddress method (Locale), 311
gGetDisplayCountry method (Locale), 311
gGetDisplayLanguage method (Locale), 311
gGetDisplayName method
  of FeatureDescriptor, 757
  of Locale, 310–311, 313
  of TimeZone, 327
gGetDocument method (DocumentEvent), 467
gGetDocumentElement method (Document), 100, 111
gGetDocumentFilter method (AbstractFormatter), 470, 483
gGetDoXxx methods (URLConnection), 212, 220
gGetDropAction method (TransferSupport), 706
gGetDropLocation method (TransferSupport), 700–701, 706
gGetDropPoint method
  (TransferHandler.DropLocation), 706
gGetElementAt method (ListModel), 371, 375
gGetEngineByXxx, getEngineFactories methods
  (ScriptEngineManager), 894–895
gGetEntry method (ZipFile), 36
gGetErrorCode method (SQLException), 256–257
gGetErrorStream method (HttpURLConnection), 227, 229
gGetExceptionWriter method (ScriptContext), 898
gGetExceptionListener method
  of Encoder, 801
  of XMLDecoder, 801
gGetExpiration method (URLConnection), 213, 216, 222
gGetExtensions method (ScriptEngineFactory), 895
gGetFieldID function (C), 1006, 1010
gGetFields method (Class), 456
gGetFileForOutput method
  (ForwardingJavaFileManager), 913
gGetFileName method (Path), 59
gGetFilePointer method (RandomAccessFile), 28, 33
gGetFileSuffixes method (ImageReaderWriterSpi), 618
gGetFillsViewportHeight method (JTable), 386
gGetFirstChild method (Node), 103, 112
gGetFocusLostBehavior method
  (FormattedTextField), 469, 483
gGetFontRenderContext method (Graphics2D), 591
gGetForeground method (JList), 378, 381
gGetFormatNames method (ImageReaderWriterSpi), 618
gGetFragment method (URI), 211
gGetFrameIcon method (JInternalFrame), 542
gGetHeaderField method (URLConnection), 213, 222
gGetHeaderFieldKey method (URLConnection), 213–215, 221
gGetHeightFields method (URLConnection), 213, 215, 221
gGetHeight method
  of ImageReader, 611, 618
  of PageFormat, 639, 646
gGetHost method (URI), 211
gGetHostXxx methods (InetAddress), 193
gGetHumanPresentableName method (DataFlavor), 680
gGetIcon method (Introspector), 755–756
gGetIconAt method (JTabbedPane), 523
gGetID method (TimeZone), 327
gGetIdentifier method (Entry), 407
gGetModifiedSince method (URLConnection), 221
getimage method (TrayIcon), 724
getImageableXxx methods (PageFormat), 640, 646
getImageURL method (SplashScreen), 713
getImageXxxByXxx methods (ImageIO), 609, 616–617
getIndex method
  of DataTruncation, 258
  of IndexedPropertyChangeEvent class, 752
  of JList.DropLocation, 701, 706
  of JTextComponent.DropLocation, 707
getIndexedXxxMethod methods
  (IndexedPropertyDescriptor), 758
getInputChild method (TreeModel), 454, 461
getInputStream method
  of Socket, 189–190, 194
  of URLConnection, 213, 222, 225, 227
  of ZipFile, 36
getInstance method
  of AlphaComposite, 595, 601
  of Cipher, 881, 886
  of Collator, 335
  of Currency, 318–319
  of KeyGenerator, 887
  of Locale, 329
  of MessageDigest, 860–862
getIntegerInstance method (NumberFormat), 470
getInterface method (Invocable), 900
getInvalidCharacters method (MaskFormatter), 485
getJavaFileForOutputXXX methods (ForwardingJavaFileManager), 909, 914
getJavaFileObjectsFromXXX methods (StandardJavaFileManager), 912
getJavaInitializationString method (PropertyEditor), 763–764, 770
getJDBCXXXVersion methods (DatabaseMetaData), 296
getKeys method (ResourceBundle), 346
getKind method (Diagnostic), 912
getLanguage method (Locale), 311
getLastChild method (Node), 104, 112
getLastModified method (URLConnection), 213, 216, 222
getLastPathComponent method (TreePath), 432–433, 439
getLastSelectedPathComponent method (JTree), 433, 439
getLayoutOrientation method (JList), 369
getLeading method (TextLayout), 592
getLength method
  of Attributes, 155
  of Document, 466
  of NamedNodeMap, 112
  of NodeList, 101, 112
getLineNumber method
  of Diagnostic, 912
  of SAXParseException, 121
getListCellRendererComponent method
  (ListCellRenderer), 378–381
getLocale method (MessageFormat), 337
getLocalHost method (Socket), 192–193
getLocalName method
  of Attributes, 155
  of Node, 149
  of XMLStreamReader, 159
getLocation method (CodeSource), 826
getMaxConnections method (DatabaseMetaData), 296
getMaxmethod (JProgressbar), 513
getMaxStatements method (DatabaseMetaData), 255, 296
getMessage method (Diagnostic), 912
getMetadata method
  of Connection, 286, 295
  of ResultSet, 287, 296
getMethodCallSyntax method (ScriptEngineFactory), 899
GetMethodID function (C), 1017–1018
getMimeType method (DataFlavor), 679
getMimeTypes method (ScriptEngineFactory), 895
getMIMETypes method (ImageReaderWriterSpi), 618
getMinimum method (JProgressbar), 513
getMinimumXXXDigits, getMaximumXXXDigits
  methods (NumberFormat), 317
getMnemonicAt method (JTabbedPane), 524
getModel method
  of Entry, 407
  of JList, 377
getMoreResults method (Statement), 273
getName method
  of Attribute, 671
  of FeatureDescriptor, 757
  of Field, 456
  of NameCallback, 857
  of NameClassPair, 965
  of Permission, 837, 842
  of Principal, 849
  of PrintService, 660
  of UnixPrincipal, 843
  of XMLStreamReader, 159
  of ZipEntry, 35
  of ZipFile, 36
getNames method (ScriptEngineFactory), 895
getNamespaceURI method (Node), 149
getNewValue method (PropertyChangeEvent), 543, 746, 752
getNextEntry method (ZipInputStream), 33–34
getNextException method (SQLException), 256–257
getNextSibling method (Node), 104, 112
getNextValue method (AbstractSpinnerModel), 487–488, 494
getNextWarning method (SQLWarning), 258
getNodeName method (Node), 104, 112, 149
getNodeValue method (Node), 104, 112
getNumberOfImages method (ImageReader), 610–611, 617
getNumberOfThumbnails method (ImageReader), 611, 618
getObject method (ResourceBundle), 346
GetObjectArrayElement function (C), 1020, 1023
GetObjectClass function (C), 59
GetObjectArrayElement method (ResourceBundle), 346
GetObjectArrayElement method (C), 1006–1007
GetOldValue method (PropertyChangeEvent), 746, 752
getOrientation method (PageFormat), 647
getOriginatingProvider method
  of ImageReader, 609, 618
  of ImageWriter, 619
getOutline method, 590
getOutputStream method (Cipher), 886
getOutline method (Cipher), 886
getOutputStream method
  of Socket, 190, 194
  of URLConnection, 213, 222, 225
getOverwriteMode method (DefaultFormatter), 483
getOwner method (Files), 63
getOwner method (Banner), 648
getParameter method (DataTruncation), 258
getParent method
  of ClassLoader, 815
  of Path, 59
  of TreeNode, 439, 441
getParentNode method (Node), 112
getPassword method
  of PasswordCallback, 858
  of RowSet, 284
getPath method
  of FileSystem, 68
  of JTree.DropLocation, 701, 707
  of TreeSelectionEvent, 453
  of URI, 211
getPath method (TreeSelectionEvent), 447, 453
getPathToRoot method (DefaultTreeModel), 434
getPercentInstance method (NumberFormat), 311, 317, 472
getPersistenceDelegate method (Encoder), 801
getPixel, getPixels methods (Raster), 620, 625
getPlaceholder, getPlaceholderCharacter methods
  (MaskFormatter), 485
getPointCount method (ShapeMaker), 560
getPopupMenu method (TrayIcon), 724
getPort method (URI), 211
getPreferredSize method (JComponent), 378–379, 466
getPreviousSibling method (Node), 112
getPreviousValue method (AbstractSpinnerModel), 487–488, 494
getPrintable method (Book), 659
getPrinterJob method (PrinterJob), 637, 646
getPrintService method
  (StreamPrintServiceFactory), 664
getPrompt method
  of NameCallback, 857
  of PasswordCallback, 858
getPropertyChange method
  (PropertyVetoException), 753
getPropertyChangeListener method
  implementing, 745
  of Component class, 746
  of PropertyChangeListener class, 751
getPropertyDescriptors method (BeanInfo), 754–756, 762
  overriding, 758–759
getPropertyNaming method (PropertyChangeEvent), 543, 746, 752
getPropertyType method (PropertyDescriptor), 757
getProtectionDomain method (Class), 825
getPrototypeCellValue method (JList), 375
getQName method (Attributes), 155
getQuery method (URI), 211
getRaster method (BufferedImage), 620, 625
getReader method (ScriptContext), 898
getReaderXXX methods (ImageIO), 609, 617
getReadMethod method (PropertyDescriptor), 757
getReadMethod method (URLConnection), 221
getRepresentationClass method (DataFlavor), 680
getRequests method (URLConnection), 221
getResultSet method (Statement), 254
getReturnAuthorization method (ServiceCenter), 979
getRGB method
  of Color, 626
  of ColorModel, 622, 626
getRoot method
  of Path, 59
  of TreeModel, 104, 454–456, 461
getRotateInstance method (AffineTransform), 586, 588
getRow method
  of JTable.DropLocation, 707
  of ResultSet, 276, 280
getRowCount method (TableModel), 386, 388, 390
getRowSelectionAllowed method (JTable), 405
getSavepointXxx methods (Savepoint), 299
setScaleInstance method (AffineTransform), 586, 588
getscheme, getschemeSpecificPart methods (URI), 211
gSelectedComponent method (JTabbedPane), 523
gSelectedIndex method (JTabbedPane), 520, 523
gSelectedValue method (JList), 366, 370
gSelectedValues method (JList), 370
gSelectedValuesList method (JList), 367
gSelectionMode method (JList), 369
gSelectionModel method (JTable), 404
gSelectionPath method (JTree), 433, 438, 447, 452
getSelectionPaths method (JTree), 446, 452
gSelectionXxx methods (JList), 378, 381
get/set naming pattern, 741–744
gShearInstance method (AffineTransform), 586, 588
gShortDescription method (FeatureDescriptor), 757
gSize method
  of ListModel, 371, 375
  of ZipEntry, 35
gSource method (Diagnostic), 912
gSourceActions method (TransferHandler), 696, 698
gSourceDropActions method (TransferSupport), 706
getSplashScreen method (SplashScreen), 709, 713
gSQLState method (SQLException), 256–257
gSQLStateType method (DatabaseMetaData), 256
gStandardFileManager method (JavaCompiler), 911
GetStaticFieldID, GetStaticXxxField functions
  (C), 1009–1010
getStrength method (Collator), 335
gString method
  of JProgressBar, 513
  of ResourceBundle, 344, 346
gStringArray method (ResourceBundle), 346
getStringChars function (C), 1002
getStringLength function (C), 1002
getStringRegion function (C), 1001–1002
getStringUTFChars function (C), 1002–1003, 1038
getStringUTFLength function (C), 1001–1002
getStringUTFRegion function (C), 1001–1002
getStringValue method (Entry), 407
getSubject method (LoginContext), 848
gSubString method (Clob), 271
getSuperclass function (C), 1050
getSymbol method (Currency), 319
gSystemClassLoader method (ClassLoader), 815
gSystemClipboard method (Toolkit), 674, 677
gSystemJavaCompiler method (ToolProvider), 907
gSystemTray method (SystemTray), 720, 723
gTabComponentAt method (JTabbedPane), 519, 524
getTabCount method (JTabbedPane), 523
gTabLayoutPolicy method (JTabbedPane), 523
gTableCellEditorComponent method
  (TableCellEditor), 411, 413, 420
gTableCellRendererComponent method
  (TableCellRenderer), 408, 419
gTableName method (CachedRowSet), 285
gTables method (DatabaseMetaData), 286, 295
gTagName method (Element), 101, 111, 149
gTags method (PropertyEditor), 762–763, 770
gTask method (JavaCompiler), 908, 912
gText method
  of Document, 466
  of XMLStreamReader, 159
gTime method
  of DateFormat, 319, 326, 472
gTimeZone method (DateFormat), 327
gTitleAt method (JTabbedPane), 523
gTooltip method (TrayIcon), 724
gTransferable method (TransferSupport), 700
gTransferData method (Transferable), 678
gTransferDataFlavors method (Transferable), 680
gTransferSize method (DataTruncation), 258
gTranslateInstance method (AffineTransform), 586, 588
gTrayIconSize method (SystemTray), 723
gTreeCellRendererComponent method
  (TreeCellRenderer), 443–445
gType method
  of Field, 456
  of ResultSet, 276, 279
gUpdateCount method (Statement), 254, 273
gURI method (Attributes), 155
gURL method
  of HyperlinkEvent, 497, 501
  of RowSet, 284
gUseCaches method (URLConnection), 221
gUserDropAction method (TransferSupport), 706
gUserInfo method (URI), 211
getUsername method (RowSet), 284
getValidCharacters method (MaskFormatter), 484
getValue method
  of AbstractSpinnerModel, 487, 494
  of Attributes, 155
  of Copies, 667
  of Entry, 407
  of JFormattedTextField, 468, 482
  of JProgressBar, 513
  of JSpinner, 485, 492
  of PropertyEditor, 769
getValueAt method (TableModel), 386–387, 390
getValueContainsLiteralCharacters method (MaskFormatter), 485
getValueCount method (Entry), 407
getValueIsAdjusting method (ListSelectionEvent), 366
getVendorName, getVersion methods
  (IIOServiceProvider), 609, 618
getVetoableChangeListener methods
  (VetoableChangeSupport), 753
getVisibleRowCount method (JList), 369
getWarnings method
  of Connection, 258
  of ResultSet, 258
  of Statement, 258
getWidth method
  of ImageReader, 611, 618
  of PageFormat, 639, 646
getWriteMethod method (PropertyDescriptor), 757
getWriter method (ScriptContext), 898
getWriterFormats helper method, 610
getWriterXXX methods (ImageIO), 609, 617
getXXX methods
  of ByteBuffer, 70, 76
  of ResultSet, 253–254
GetXXXArrayElements functions (C), 1020, 1023
GetXXXArrayRegion functions (C), 1022–1023
GetXXXField functions (C), 1006, 1010, 1038
GIF image format, 608
  animated, 610, 708
  image manipulations on, 630
  printing, 659
  writing to, not supported, 608
GlassFish server, 302
Glob patterns, 65–66
GMail, 231
Gnome desktop environment
  drag and drop in, 691
  supporting API operations in, 714
Gnu C compiler, 993–994
Gödel’s theorem, 816
GradientPaint class, 581–582
  constructor, 581, 583
  cyclic parameter, 582
grant keyword, 828, 844, 875
Graphic Java™ (Geary), 382, 421
Graphics class, 549
drawXXX, fillXXX methods, 553
get/setClip methods, 589–591, 639
Graphics2D class, 549–724
clip method, 551, 589–592, 639
draw method, 551–553, 571
fill method, 551–552, 571
getContext method, 591
rotate method, 584, 589
scale method, 584, 589
setComposite method, 551, 594, 601
setPaint method, 551, 581, 583
setRenderingHint, setRenderingHints methods, 551, 601–604, 607
setStroke method, 551, 572, 580
setTransform method, 587, 589
shear method, 584, 589
transform method, 551, 587, 589
translate method, 584, 589, 648
Grid bag layout, 125–129
GridBagConstraints class, 126
GridBagLayout class, 125–139
GridBagPane class, 129
Groovy programming language, 894–895, 902
group, groupCount methods (Matcher), 87, 92
GSS-API, 892
&gt; entity reference (HTML, XML), 98
GUI (Graphical User Interface)
  scripting events for, 901–906
  tabbed panes in, 772
GUI-based property editors, 765–770
Guide to the SQL Standard, A (Date/Darwen), 239
H
H, in masks, 474
Half-closing connections, 201–202
Handbook of Applied Cryptography, The
  (Menezes et al.), 863
handle method (CallbackHandler), 857
handleGetObject method (ResourceBundle), 346
Handles (in trees), 424, 442
HashAttributeSet class, 665
hashCode method
  and remote objects, 980
of Annotation, 930
of Permission, 834
hash/Digest.java, 861
HashDocAttributeSet class, 665
HashPrintJobAttributeSet class, 666
HashPrintRequestAttributeSet class, 637, 665
HashPrintServiceAttributeSet class, 666
Haskell programming language, 894
hasMoreElements method (Enumeration), 1036, 1038
hasNext method (XMLStreamReader), 158
hasRemaining method (Buffer), 74
Header information, from server, 212
Headers (Swing tables)
   rendering, 409
   scrolling, 383
helloNative/HelloNative.c, 993
helloNative/HelloNative.h, 992
helloNative/HelloNative.java, 991
helloNative/HelloNativeTest.java, 994
hex editor, modifying bytecodes with, 819
Hidden commands, in XML comments, 99
HORIZONTAL_SPLIT value (JSplitPane), 515
HORIZONTAL_WRAP value (JList), 365
Hosts
   and IP addresses, 192–193
   local, 192
Hot deployment, 808
HTML (HyperText Markup Language)
   attributes in, 98
   displaying with JEditorPane, 463, 494–501
   end and empty tags in, 96
   forms in, 223
   generating from XML files, 173–176
   mixing with JSP, 913
   printing, 659
   vs. XML, 95–96
HTMLDocument class, 463
HTTP (Hypertext Transfer Protocol), 238
   request headers in, 214
   using SSL over, 892
HTTP request headers, 214
http: URL prefix, 210, 495, 828
https: URL prefix, 210, 892
HttpURLConnection class
   getErrorStream method, 227, 229
HyperlinkEvent class
   getURL method, 497, 501
HyperlinkListener interface
   hyperlinkUpdate method, 496, 501
Hyperlinks, 495

I (int), type code, 44, 1011
IANA Character Set Registry, 20
IBM, 93, 100
IBM DB2, 245
ICC profiles, 621
Icons
   for beans, 755
   for frames, 525
   in column headers, 409
   in table cells, 408
   in trees, 429–430, 442
   loading, 755
   one-touch expand, 515
   tray, 719
ID, IDREF, IDREFS attribute types (DTDs),
   117–118
Identity (do-nothing) transformation, 163
IDL (Interface Definition Language), 956
IDs, uniqueness of, 118, 128
IETF BCP 47, 309–310
II0Image class, 611
   constructor, 619
II0ServiceProvider class
   getVendorName, getVersion methods, 609, 618
   IllegalAccessException, 456
   IllegalArgumentException, 320, 487, 492, 494, 1025
   IllegalStateException, 611, 701
Imageable area, 640
imageFlavor constant (DataFlavor), 680–681
ImageIcon class, using JavaBeans persistence
   for, 784
ImageInputSteam class, “seek forward only” of,
   610
ImageI0 class
   createImageXxxStream methods, 610, 617
   determining image type, 608
   getI mageXxxByXxx methods, 609, 616–617
   getReaderXxx, getWriterXxx methods, 609, 617
   read, write methods, 608, 616
imageI0/ImageI0Frame.java, 613
ImageOutputStream class, 611
imageProcessing/ImageProcessingFrame.java, 630
ImageReader class, 608
   getHeight method, 611, 618
   getNumImages method, 610–611, 617
   getNumThumbnails method, 611, 618
   getOriginatingProvider method, 609, 618
   getWidth method, 611, 618
   read method, 617
ImageReader class (cont.)
    readThumbnail method, 618
    setInput method, 617
ImageReaderWriterSpi class
    getFileSuffixes method, 618
    getFormatNames method, 618
    getMIMETypes method, 618
Images
    blurring, 629
    buffered, 582
    types of, 620
    color-model-specific values of, 621
    constructing from pixels, 619–626
    converting between Java and native formats, 681
    edge detection of, 629
    filtering, 626–636
    getting size of, before reading, 611
    incremental rendering of, 619
    manipulating, 619–636
    metadata in, 611
    multiple, in a file, 610–619
    reading, 610–611
    writing, 611
    printing, 637–647, 659, 662
    readers for, 608–619
    service provider interface of, 609
    rotating, 603, 627–628
    scaling, 603–604
    superimposing, 592
    thumbnails for, 611
    transferring via clipboard, 680
    writers for, 608–619
    supported types in, 609
ImageTransferable class, 680–681, 696
imageTransfer/imageTransferFrame.java, 683
ImageViewerBean class, 728–745, 754
    fileName property, 744
ImageViewerBeanInfo class, 754–755
imageViewer/imageViewerBean.java, 729
ImageWriter class, 608, 611
    canInsertImage method, 612, 619
    getOriginatingProvider method, 619
    setOutput method, 618
    write method, 611, 618
    writeInsert method, 612, 619
IMAP (Internet Message Access Protocol), 892
#IMPLIED attribute default (DTDs), 118
implies method
    of Permission, 834–836, 842
    of ProtectionDomain, 826
import statement, 808
importData method (TransferHandler), 699–701, 705
INCLUDE environment variable, 1033
include method (RowFilter), 397, 407
Incremental rendering, 619
Indexed color model, 629
Indexed properties, 744–745
IndexedPropertyChangeEvent class
    constructor, 752
    getIndex method, 752
IndexedPropertyDescriptor class
    constructor, 758
    getIndexedXxx methods, 758
indexOfComponent, indexOfTab methods
    (JTabbedPane), 518, 523
indexOfTabComponent method (JTabbedPane), 524
IndexOutOfBoundsException, 610–611
InetAddress class, 192–193
    getAddress method, 192–193
    getByName method, 192–193
    getHostXxx methods, 193
    getLocalHost method, 192–193
inetAddress/ InetAddressTest.java, 192
InetSocketAddress class
    constructor, 209
    isUnresolved method, 209
Infinite trees, 457
Inheritance trees, 256
@Inherited annotation, 934
Inherited interface, 931
init method
    of Cipher, 886
    of KeyGenerator, 887
InitialContext class, 963
Initialization blocks, for shared libraries, 996
initialize method
    of DefaultPersistenceDelegate, 787, 802
    of LoginModule, 858
    of PersistenceDelegate, 802
    of SimpleLoginModule, 850
Input fields. See Text fields
Input streams, 2
    as input source, 100
    keeping open, 201
    monitoring, 509–514
    total number of bytes in, 510
Input validation mask, 467
InputSource class, 121
InputStream class, 2–4, 7
   available method, 2, 4, 510
   close method, 3–4
   mark, markSupported methods, 4
   read method, 2–3
   reset method, 4
   skip method, 3
INPUT_STREAM data source (print services), 661
InputStreamReader class, 13, 509
InputVerifier class
   verify method, 471
insert keyword, 836
INSERT statement (SQL), 244
   and autogenerated keys, 274
   executing, 252, 254, 269
   in batch updates, 298
   vs. methods of ResultSet, 278
insertNodeInto method (DefaultTreeModel), 433, 439
insertRow method (ResultSet), 278, 280
insertString method (DocumentFilter), 470–471, 483–484
insertTab method (JTabbedPane), 518, 523
insertUpdate method (DocumentListener), 464, 467
Insets class, using JavaBeans persistence for, 784
Inside Java™ 2 Platform Security (Gong et al.), 804
installUI method (LayerUI), 547
Instance fields
   accessing from native code, 1005–1009
   vs. properties, 728
instanceof operator, 979
instantiate method
   of DefaultPersistenceDelegate, 802
   of PersistenceDelegate, 785–787, 802
int type
   printing, 14
   storing, 25
   type code for, 44, 1011
   vs. C types, 997
   writing in binary format, 25
IntBuffer class, 77
INTEGER (INT) data type (SQL), 245, 301
Integers
   supported locales for, 468
   validating input of, 468
IntegerSyntax class, 667
@interface declaration, 920, 926
Interfaces
   accessing script classes with, 900
   annotating, 931
   implementing in script engines, 899
   shared, 959–961
Internal frames, 524–534
   cascading/tiling, 527–531
   closing, 531
   dialogs in, 533
   dragging, 534
   grabbers of, 524
   icons for, 525
   managing on desktop, 534
   positioning, 527
   resizable, 528
   selected, 527
   states of, 527–528
   visibility of, 526
internalFrame/DesktopFrame.java, 535
InternalFrameListener interface
   internalFrameClosing method, 533
International currency character, 20
InternationalFormatter class, 470
Internationalization, 305–362
Interpolation, 627
   for gradients, 581
   strategies of, 627
   when transforming images, 603–604, 627
Interruptible sockets, 202–209
interruptible/InterruptibleSocketTest.java, 203
intersect method (Area), 571–572
Introspector class
   getBeanInfo method, 755–756
   intValue method (Number), 312
   Invalid pointers (C, C++), 990
   InvalidPathException, 57
   InverseEditor class, 759
   Invocable interface, 899
   getInterface method, 900
   invokeXxx methods, 899–900
Invocation API, 1028–1034
invocation/InvocationTest.c, 1030
invokeFunction method (Invocable), 899
IOException, 189, 498
IP addresses, 187, 192–193
   validating, 474–476
IPP (Internet Printing Protocol) 1.1, 671
IPv6 addresses, 192
isAfterLast method (ResultSet), 277, 280
isAnnotationPresent method (AnnotatedElement), 926
IsAssignableFrom function (C), 1037, 1050
isBeforeFirst method (ResultSet), 277, 280
isCanceled method (ProgressMonitor), 506, 514
isCellEditable method
  of AbstractCellEditor, 412
  of AbstractTableModel, 410
  of CellEditor, 420
  of DefaultTableModel, 410
  of TableModel, 390, 410
isCharacters method (XMLStreamReader), 159
isCloseable method (JInternalFrame), 541
isClosed method
  of JInternalFrame, 542
  of ResultSet, 255
  of Socket, 191
  of Statement, 254
isConnected method (Socket), 191
isContinuousLayout method (JSplitPane), 517
isDataFlavorAvailable method (Clipboard), 678
isDataFlavorSupported method (Transferable), 678
isDaylightTime method (TimeZone), 328
isDesktopSupported method (Desktop), 714, 719
isDirectory method
  of BasicFileAttributes, 64
  of ZipEntry, 35
isDrop method (TransferSupport), 701, 706
isEchoOn method (PasswordCallback), 858
isEditValid method (JFormattedTextField), 469, 471, 483
isEndElement method (XMLStreamReader), 159
isExecutable method (Files), 63–64
isExpert method (FeatureDescriptor), 757
isFirst method (ResultSet), 277, 280
isGroupingUsed method (NumberFormat), 317
isHidden method
  of FeatureDescriptor, 757
  of Files, 63–64
isIcon method (JInternalFrame), 528, 542
isIconifiable method (JInternalFrame), 541
isIgnoringElementContentWhitespace method
  (DocumentBuilderFactory), 122
isImageAutoSize method (TrayIcon), 724
isIndeterminate method (JProgressBar), 513
isInputShutdown method (Socket), 202
isInsert method (JList.DropLocation), 701, 706
isInsertColumn, isInsertRow methods (JTable.DropLocation), 707
isLast method (ResultSet), 277, 280
isLeaf method
  of DefaultTreeModel, 428
  of TreeModel, 431, 454, 461
  of TreeNode, 428, 430
isLenient method (DateFormat), 326
isMaximizable method (JInternalFrame), 541
isMaximum method (JInternalFrame), 542
isMimeTypeEqual method (DataFlavor), 679
isNamespaceAware method
  of DocumentBuilderFactory, 149
  of SAXParserFactory, 154
ISO (International Organization for Standardization), 307
ISO 216 paper sizes, 345
ISO 3166–1 country codes, 307–308
ISO 4217 currency identifiers, 318
ISO 639–1 language codes, 307–308
ISO 8601 format, 271, 932
ISO 8859 Alphabet Soup, The, 21
ISO 8859–1 encoding, 13, 20–21
ISO 8859–15 encoding, 20
isOneTouchExpandable method (JSplitPane), 517
isOutputShutdown method (Socket), 202
isPaintable method (PropertyEditor), 762, 770
  overriding, 766
isParseIntegerOnly method (NumberFormat), 317
isQViewer, 287
isReadable method (Files), 63–64
isRegularFile method
  of BasicFileAttributes, 64
  of Files, 63–64
isResizable method (JInternalFrame), 541
isSelected method (JInternalFrame), 530, 542
is/set naming pattern, 742
isShared method (FileLock), 80
isStartElement method (XMLStreamReader), 159
isStringPainted method (JXStreamReader), 513
isSupported method
  of Desktop, 714, 719
  of SystemTray, 719, 723
isSymbolicLink method
  of BasicFileAttributes, 64
  of Files, 63–64
isUnresolved method (InetSocketAddress), 209
isValidating method
  of DocumentBuilderFactory, 122
  of SAXParserFactory, 154
isVisible method (JInternalFrame), 542
isWhiteSpace method (XMLStreamReader), 159
isWritable method (Files), 63–64
item method
  of NodeList, 112
  of NodeList, 101, 112, 120
Iterable interface, 65
Iterator interface, 252
iterator method (SQLException), 256–257
J
J (long), type code, 44, 1011
JAAS (The Java Authentication and Authorization Service), 842–858
configuration files in, 843, 847
login modules in, 844, 849–858
jaas/jaas.config, 857
jaas/JAASTest.java, 856
jaas/JAASTest.policy, 857
jaas/SimpleCallbackHandler.java, 855
jaas/SimpleLoginModule.java, 853
jaas/SimplePrincipal.java, 852
JAR files
automatic registration in, 248
class loaders in, 806
code base of, 822
for plugins, 806
manifest of, 34, 708
packaging beans in, 731–733
resources in, 342
signing, 867–868, 873–878
jar: URL prefix, 210
JarInputStream class, 34
JarOutputStream class, 34
jarray type (C++), 1020, 1037
jarsigner program, 867–868, 874
Java 2D API, 549–724
features supported in, 550
rendering pipeline, 550–552
Java EE (Enterprise Edition), 238
Java Plug-in, 873
java program
-javaagent option, 950
jdbc.drivers property in, 248
-noverify option, 820
port number in, 967
security managers in, 828
-splash option, 708
Java programming language
deployment directory of, 875–876
internationalization support in, 305
platform-independent, 25
security of, 822–826
types, vs. C types, 997
using with CORBA/SOAP, 956
vs. SQL, 265
Java™ Virtual Machine Specification, The (Lindholm/Yellin), 819
java.awt.AlphaComposite API, 753
java.awt.datatransfer API, 674
java.awt.datatransfer.Clipboard API, 677–678, 680, 689
java.awt.datatransfer.ClipboardOwner API, 678
java.awt.datatransfer.DataFlavor API, 679
java.awt.datatransfer.FlavorListener API, 680
java.awt.datatransfer.Transferable API, 678, 680
java.awt.Desktop API, 719
java.awt.dnd package, 691
java.awt.font.TextLayout API, 591–592
java.awt.geom API, 49
java.awt.geom.AffineTransform API, 587–589
java.awt.geom.Arc2D.Double API, 569
java.awt.geom.Area API, 572
java.awt.geom.CubicCurve2D.Double API, 569
java.awt.geom.GeneralPath API, 570
java.awt.geom.Path2D API, 570
java.awt.geom.Path2D.Float API, 570
java.awt.geom.QuadCurve2D.Double API, 569
java.awt.geom.RoundRectangle2D.Double API, 569
java.awt.GradientPaint API, 583
java.awt.Graphics API, 591
java.awt.Graphics2D API, 552, 580, 583, 589, 591, 601, 607
java.awt.image.AffineTransformOp API, 635
java.awt.image.BufferedImage API, 625
java.awt.image.BufferedImageOp API, 635
java.awt.image.ByteLookupTable API, 635
java.awt.image.ColorModel API, 626
java.awt.image.ConvolveOp API, 636
java.awt.image.Kernel API, 636
java.awt.image.LookupOp API, 635
java.awt.image.Raster API, 625
java.awt.image.RescaleOp API, 635
java.awt.image.ShortLookupTable API, 636
java.awt.image.WritableRaster API, 626
java.awt.print.Book API, 659
java.awt.print.PageFormat API, 646–647
java.awt.print.Printable API, 645
java.awt.print.PrinterJob API, 646, 659
java.awt.RenderingHints API, 608
java.awt.MultipleSelectionModel API, 617
java.awt.TextArea API, 713
java.awt.SystemTray API, 723
java.awt.TexturePaint API, 583
java.awt.Toolkit API, 677
java.awt.TrayIcon API, 724
JavaBeans, 725–802
JavaBeans persistence, 779–802
advantages of, 792
complete example for, 791–802
JavaBeans persistence (cont.)
delegates for, 784–786, 788
for arbitrary data, 784–791
vs. serialization, 779
java.beans.BeanDescriptor API, 772
java.beans.BeanInfo API, 756, 762, 771
java.beans.Customizer API, 779
java.beans.DefaultPersistenceDelegate API, 802
java.beans.Encoder API, 801
java.beans.ExceptionListener API, 801
java.beans.Expression API, 802
java.beans.FeatureDescriptor API, 757
java.beans.IndexedPropertyChangeEvent API, 752
java.beans.IndexedPropertyDescriptor API, 758
java.beans.Introspector API, 756
java.beans.PersistenceDelegate API, 802
java.beans.PropertyChangeEvent API, 543, 752
java.beans.PropertyChangeListener API, 751
java.beans.PropertyChangeSupport API, 751
java.beans.PropertyDescriptor API, 757, 761
java.beans.PropertyEditor API, 769
java.beans.PropertyVetoException API, 543, 753
java.beans.SimpleBeanInfo API, 756
java.beans.Statement API, 802
java.beans.VetoableChangeListener API, 543, 752
java.beans.VetoableChangeSupport API, 752­753
java.beans.XMLDecoder API, 801
java.beans.XMLEncoder API, 801
javac program
-encoding option, 341
-processor option, 937
XprintRounds option, 939
JavaCompiler class
getStandardFileManager method, 911
getTask method, 908, 912
Javadoc, 934
javaFileListFlavor constant (DataFlavor), 700
JavaFileManager class, 908
customizing, 909
JavaFileObject interface, 908
implementing, 909
javah program, 992, 1039
JavaHelp, 494
java.io.BufferedInputStream API, 12
java.io.BufferedOutputStream API, 12
java.io.Closeable API, 8
java.io.DataInput API, 27
java.io.DataOutput API, 28
java.io.File API, 60
java.io.FileInputStream API, 11, 74
java.io.FileOutputStream API, 11, 74
java.io.File.separator API, 9
java.io.Flushable API, 8
java.io.InputStream API, 3–4
java.io.ObjectInputStream API, 41–42
java.io.ObjectOutputStream API, 41
java.io.OutputStream API, 4
java.io.PrintWriter API, 15–16
java.io.PushbackInputStream API, 12
java.io.RandomAccessFile API, 32, 74
java.lang.annotation package, 931
java.lang.annotation.Annotation API, 930
java.lang.Appendable API, 8
java.lang.CharSequence API, 8–9
java.lang.Class API, 815, 825
java.lang.ClassLoader API, 815
java.lang.Readable API, 8
java.lang.reflect.AnnotatedElement API, 926
java.lang.SecurityManager API, 825
java.lang.System API, 996
java.lang.Thread API, 815
JavaMail, 216, 230
java.net package
socket connections in, 190
supporting IPv6 addresses, 192
URLs vs. URIs in, 210
java.net.HttpURLConnection API, 229
java.net.InetAddress API, 193
java.net.InetSocketAddress API, 209
java.net.ServerSocket API, 197
java.net.Socket API, 190–191, 202
java.net.URL API, 220
java.net.URLClassLoader API, 815
java.net.URLConnection API, 220–222
java.net.URLDecoder API, 230
java.net.URLEncoder API, 229
java.nio package
customizing, 909
implementing, 909
java.nio.charset.Charset API, 24
java.nio.file.attribute.BasicFileAttributes API, 64
java.nio.file.Files API, 61–64, 67
javax.security.auth.login.LoginContext API, 847–848
d javax.security.auth.spi>LoginModule API, 858
d javax.security.auth.Subject API, 848
d javax.sql package, 302
d javax.sql.rowset package, 282
d javax.sql.RowSet API, 284–285
d javax.sql.rowset.CachedRowSet API, 285
d javax.sql.rowset.RowSetFactory API, 286
d javax.sql.rowset.RowSetProvider API, 285
d javax.swing.AbstractSpinnerModel API, 494
d javax.swing.CellEditor API, 420
d javax.swing.DefaultCellEditor API, 419
d javax.swing.DefaultListModel API, 377
d javax.swing.DefaultRowSorter API, 406
d javax.swing.event.DocumentEvent API, 467
d javax.swing.event.DocumentListener API, 467
d javax.swing.event.HyperlinkEvent API, 501
d javax.swing.event.HyperlinkListener API, 501
d javax.swing.event.ListSelectionListener API, 462
d javax.swing.event.TreeMouseListener API, 462
d javax.swing.event.TreeSelectionEvent API, 453
d javax.swing.event.TreeSelectionListener API, 452
d javax.swing.ColorChooser API, 696
d javax.swing.JComponent API, 431, 466, 543, 695, 753
d javax.swing.JDesktopPane API, 541
d javax.swing.JEditorPane API, 500
d javax.swing.JFileChooser API, 696
d javax.swing.FormattedTextField API, 482–483
d javax.swing.FormattedTextField.AbstractFormatter API, 483
d javax.swing.JFrame API, 705
d javax.swing.JLabel API, 541–542
d javax.swing.List API, 369–370, 375, 377, 381, 696, 705
d javax.swing.ListDropLocation API, 706
d javax.swing.JProgressBar API, 512–513
d javax.swing.JSpinner API, 492
d javax.swing.JSpinner.DateEditor API, 494
d javax.swing.JSplitPane API, 517
d javax.swing.JTabbedPane API, 523–524
d javax.swing.JTable API, 386, 404–405, 419, 696, 705
d javax.swing.JTableDropLocation API, 707
d javax.swing.JTree API, 430, 438–439, 452, 696, 705
d javax.swing.JTree.DropLocation API, 707
d javax.swing.ListCellRenderer API, 381
d javax.swing.ListModel API, 375
d javax.swing.ListSelectionModel API, 406
d javax.swing.plaf.LayerUI API, 547–548
d javax.swing.ProgressMonitor API, 513–514
d javax.swing.ProgressMonitorInputStream API, 514
d javax.swing.RowFilter API, 407
d javax.swing.RowFilter.Entry API, 407
d javax.swing.SpinnerDateModel API, 493
d javax.swing.SpinnerListModel API, 493
d javax.swing.SpinnerNumberModel API, 492–493
d javax.swing.table.TableCellEditor API, 420
d javax.swing.table.TableCellRenderer API, 419
d javax.swing.table.TableColumn API, 406, 419
d javax.swing.table.TableColumnModel API, 405
d javax.swing.table.TableModel API, 390, 404
d javax.swing.table.TableRowSorter API, 406
d javax.swing.table.TableStringConverter API, 406
d javax.swing.text.DefaultFormatter API, 483
d javax.swing.text.Document API, 466–467
d javax.swing.text.DocumentFilter API, 483–484
d javax.swing.text.JTextComponent API, 696, 705
d javax.swing.text.JTextComponent.DropLocation API, 707
d javax.swing.text.MaskFormatter API, 484–485
d javax.swing.TransferHandler API, 696, 698, 705
d javax.swing.TransferHandler.DropLocation API, 706
d javax.swing.TransferHandler.TransferSupport API, 706
d javax.swing.tree.DefaultMutableTreeNode API, 431, 444
d javax.swing.tree.DefaultTreeCellRenderer API, 445
d javax.swing.tree.DefaultTreeModel API, 431, 439–440
d javax.swing.tree.MutableTreeNode API, 430
d javax.swing.tree.TreeCellRenderer API, 445
d javax.swing.tree.TreeModel API, 431, 461–462
d javax.swing.tree.TreePath API, 439
d javax.swing.tree.TreePath API, 439
d javax.tools.Diagnostic API, 912
d javax.tools.DiagnosticCollector API, 912
d javax.tools.ForwardingJavaFileManager API, 913
d javax.tools.JavaCompiler API, 911–912
d javax.tools.JavaCompiler.CompilationTask API, 912
d javax.tools.SimpleFileObject API, 913
d javax.tools.StandardJavaFileManager API, 912
d javax.tools.Tool API, 911
d javax.xml.parsers.DocumentBuilder API, 111, 121, 163
d javax.xml.parsers.DocumentBuilderFactory API, 111, 122, 149
d javax.xml.parsers.SAXParser API, 154
javax.xml.parsers.SAXParserFactory API, 154
javax.xml.stream.XMLInputFactory API, 158
javax.xml.stream.XMLOutputFactory API, 171
javax.xml.stream.XMLStreamReader API, 158–159
javax.xml.stream.XMLStreamWriter API, 172
javax.xml.transform.dom.DOMResult API, 183
javax.xml.transform.dom.DOMSource API, 164
javax.xml.transform.sax.SAXSource API, 183
javax.xml.transform.stream.StreamResult API, 164
javax.xml.transform.stream.StreamSource API, 182
javax.xml.transform.Transformer API, 164
javax.xml.transform.TransformerFactory API, 163, 182
javax.xml.xpath.XPath API, 146
javax.xml.xpath.XPathFactory API, 146
JAXP (Java API for XML Processing), 100
jboolean type (C), 997, 1022
jbooleanArray type (C), 1020
jbyte type (C), 997
jbyteArray type (C), 1020
jchar type (C), 997
jcharArray type (C), 1020
JCheckBox class, 411
jclass type (C), 1017
JColorChooser class
drag-and-drop behavior of, 692
setDragEnabled method, 696
JComboBox class, 411
JComponent class
addVetoableChangeListener method, 543, 753
attaching verifiers to, 471
extending, 378, 748
firePropertyChange method, 746
fireVetoableChange method, 748, 753
getPreferredSize method, 378–379, 466
paintComponent method, 378–379
putClientProperty method, 426, 431, 534
removeVetoableChangeListener method, 753
setPreferredSize method, 466
setTransferHandler method, 692, 695
JDBC API, 235–303
configuration of, 244–251
debugging, 250
design of, 236–239
tracing, 250
uses of, 238–239
versions of, 235
JDBC API Tutorial and Reference (Fisher et al.), 279, 301
JDBC drivers
escape syntax in, 271–272
JAR files for, 246
registering classes for, 248
scrollable/updatable result sets in, 276
types of, 236–237
JDBC/ODBC bridge, 237
JdbcRowSet interface, 282
JDesktopPane class, 525
getAllFrames method, 527–530, 541
no built-in cascading/tiling in, 527–531
setDragMode method, 534, 541
JDialog class, 533
JDK (Java Development Kit)
Apache Derby database, 246
bootstrap registry service, 961
default character encodings in, 21
DOM parser, 100
java.util.prefs.Base64 class, 216
keytool program, 865
native2ascii program, 341
policies program, 833
rmiregistry service, 967
serialver program, 52
src.jar file, 1030
StylePad demo, 463
SunJCE ciphers, 881
jdouble type (C), 997
jdoubleArray type (C), 1020
JEditorPane class, 463, 494–501
addHyperlinkListener method, 500
drag-and-drop behavior of, 692
setEditable method, 496
setPage method, 496, 500
JFileChooser class
drag-and-drop behavior of, 692
setTransferHandler method, 705
jfloat type (C), 997
jfloatArray type (C), 1020
JEditorPane class, 494–501
addHyperlinkListener method, 500
drag-and-drop behavior of, 692
setEditable method, 496
setPage method, 496, 500
JFileChooser class
drag-and-drop behavior of, 692
setTransferHandler method, 705
jfloat type (C), 997
jfloatArray type (C), 1020
JFormattedTextField class, 467–485
commitEdit method, 469, 482
constructor, 482
drag-and-drop behavior of, 692
get/setFocusLostBehavior methods, 469, 483
get/setValue methods, 468–469, 482
isEditValid method, 469, 471, 483
setText method, 469
supported formatters, 472–474
JFrame class, 524
adding transfer handler to, 699
setTransferHandler method, 705
JINI (Apache River), 961
jint type (C), 997
jintArray type (C), 1020
JInternalFrame class, 531, 533–543
    closed property, 746
    constructor, 525, 541
    getDesktopPane method, 542
    get/setContentPane methods, 542
    get/setFrameIcon methods, 525, 542
    is/setClosable methods, 541
    is/setClosed methods, 531, 542
    is/setIcon methods, 528, 542
    is/setIconifiable methods, 541
    is/setMaximum methods, 528, 542
    is/setResizable methods, 541
    is/setSelected methods, 527, 530, 542
    is/setVisible methods, 526, 542
moveToXxx methods, 542
no built-in cascading/tiling in, 527–531
reshape method, 526, 542
show method, 542
JLabel class, 737, 754
    extending, 442–443, 743
    using with JList, 379
JLayer class, 543–548
    constructor, 547
    setLayerEventMask method, 547
JList class, 364–370
    addListSelectionListener method, 369
    as generic type, 364
    configuring for custom renderers, 378
    constructor, 369, 375
    drag-and-drop behavior of, 692, 700–701
    getBackground, getForeground, getSelectionXxx methods, 378, 381
    getModel method, 377
    getSelectedValue method, 366, 370
    getSelectedValues method, 370
    getSelectedValuesList method, 367
    get/setLayoutOrientation methods, 369
    get/setPrototypeCellValue methods, 375
    get/setSelectionMode methods, 365, 369
    get/setVisibleRowCount methods, 365, 369
    HORIZONTAL_WRAP, VERTICAL, VERTICAL_WRAP values, 365
    setCellRenderer method, 378, 381
    setDragEnabled method, 696
    setDropMode method, 700, 705
    setFixedCellXxx methods, 375
    visual appearance of data vs. data storage in, 370
JList.DropLocation class
    getIndex method, 701, 706
    isInsert method, 701, 706
jlong type (C), 997
jlongArray type (C), 1020
JMX (Java Management Extensions), 787
JNDI service, 302
    class loaders in, 806
JndiLoginModule class, 844
JNI (Java Native Interface), 990–1050
    accessing
        array elements in, 1019–1023
        functions in C++, 1000
        calling convention in, 1000
        debugging mode of, 1029
        error handling in, 1023–1028
        invoking Java methods from native code in, 1012–1019
    online documentation for, 1001
JNI functions, 1006
   JNICALL macro, 992
    JNICALL function (C), 1029–1030, 1034
    JNICALL macro, 992
    jni.h file, 997
    JNI_OnLoad method, 996
    JobAttributes class (obsolete), 671
    jobject type (C), 1017, 1037
    jobjectArray type (C), 1020
Join styles, 572–573
JoinRowSet interface, 282
JOptionPane class
    showInternalXxxDialog methods, 533
    Panel class, 642
    adding layers to, 543
    extending, 743, 748
PasswordField class, 462
    drag-and-drop behavior of, 692
JPEG image format, 608
    image manipulations on, 630
    printing, 659
    reading, 609
    transparency in, 708
JProgressBar class, 501–505, 512, 710
    constructor, 512
    get/setMinimum, get/setMaximum methods, 502, 513
    get/setString methods, 502, 513
    getValue methods, 513
    is/setIndeterminate methods, 502, 513
    is/setStringPainted methods, 502, 513
    setValue methods, 502, 513
JScrollPane class
  with JList, 364, 372
  with JTabbedPane, 518–519
  with JTable, 383
JSF (JavaServer Faces), 222, 726
jshort type (C), 997
jshortArray type (C), 1020
JSP (JavaServer Pages), 726, 913–918
JSpinner class, 485–494
  constructor, 492
  getValue method, 485, 492
  setEditor method, 492
  setValue method, 492
JSplitPane class, 514–517
  constructor, 517
  HORIZONTAL_SPLIT, VERTICAL_SPLIT values, 515
  is/setContinuousLayout method, 515, 517
  is/setOneTouchExpandable methods, 515, 517
  setXxxComponent methods, 517
js.properties file, 902
JSSE (Java Secure Socket Extension), 892
jstring type (C), 1000, 1017, 1037
JTabbedPane class, 518–524, 772
  addChangeListener method, 520, 524
  addTab method, 518, 523
  constructor, 518, 523
  getSelectedComponent method, 523
  get/setComponentAt methods, 523
  get/setIconAt methods, 523
  get/setMnemonicAt methods, 518, 524
  get/setSelectedIndex methods, 518, 520, 523
  get/setTabComponentAt methods, 519, 524
  get/setTabLayoutPolicy methods, 518, 523
  get/setTitleAt methods, 523
  getTabCount method, 523
  indexOfComponent method, 518, 523
  indexOfTab method, 518, 523
  indexOfTabComponent method, 524
  insertTab method, 518, 523
  removeTab method, 518, 523
JTable class, 381–420
  addColumn method, 398, 405
  asymmetric, 390
  cell editors, automatically installed, 411
  constructor, 386
  convertXxxIndexToModel methods, 395, 405
  default rendering actions, 391
  drag-and-drop behavior of, 692, 700–701
  getColumnModel method, 404
  getDefaultValueEditor method, 419
  getSelectionModel method, 404
  get/setAutoCreateRowSorter methods, 384, 386, 395
  get/setCellSelectionEnabled method, 394, 405
  get/setColumnSelectionAllowed method, 394, 405
  get/setDefaultRenderer methods, 409, 419
  get/setFillViewportHeight method, 384, 386
  get/setRowSelectionAllowed method, 394, 405
  moveColumn method, 399, 405
  not generic, 382
  print method, 384, 386
  removeColumn method, 398, 405
  resize modes, 393
  setAutoResizeMode method, 393, 404
  setDragEnabled method, 696
  setDropMode method, 700, 705
  setRowHeight method, 393–394, 404
  setRowMargin method, 394, 404
  setRowSorter method, 395, 405
JTable.DropLocation class
  getColumn, getRow methods, 707
  isInsertColumn, isInsertRow methods, 707
JTextArea class, 462
  drag-and-drop behavior of, 692
  extending, 835
JTextComponent class
  drag-and-drop behavior of, 700–701
  setDragEnabled method, 696
  setDropMode method, 700, 705
JTextField class, 462
  drag-and-drop behavior of, 692
  with DefaultCellEditor, 411
JTextPane class, 463
  drag-and-drop behavior of, 692
JTree class, 420–462
  addTreeSelectionListener method, 445
  constructor, 421–422, 430
  drag-and-drop behavior of, 692, 700–701
  getLastSelectedPathComponent method, 433, 439
  getSelectionPath method, 433, 438, 447, 452
  getSelectionPaths method, 446, 452
  identifying nodes, 431
  makeVisible method, 434, 439
  scrollPathToVisible method, 434, 439
  setDragEnabled method, 696
  setDropMode method, 700, 705
  setRootVisible method, 428, 430
  setShowsRootHandles method, 428, 430
JTree.DropLocation class
  getChildIndex method, 707
  getPath method, 701, 707
JUnit 4 tool, 920
Just-in-time compiler, 1029
jvm pointer (C), 1029
JVM (Java virtual machine)
  bootstrap class loader in, 805
class files in, 804
  creating, 1029
embedding into native code, 1028
passing command-line arguments to, 827
terminating, 821–842, 1029
transferring object references in, 689

K
Kerberos protocol, 842, 892
Kernel class, 630, 636
Kernel, of a convolution, 629
Keyboard, reading information from, 2, 13
KeyGenerator class, 882
  generateKey method, 882, 887
  getInstance method, 887
  init method, 887
KeyPairGenerator class, 889
Keys (SQL)
  autogenerated, 273–274
  generating, 882–887
  primary, 273
keystore keyword, 874
KeyStoreLoginModule class, 844
Keystores, 865–868, 874
  referencing in policy files, 874
Keystrokes
  filtering, 467
  monitoring, 464
keytool program, 865–868
Key/Value pairs. See Properties
Kerberos protocol, 842, 892
KeyStoreLoginModule class, 844
Keystores, 865–868, 874
  referencing in policy files, 874

L
L
  (object), type code, 44, 1011
    in masks, 474
Landscape orientation, 587
Language codes, 307–308
Language tags, 309–310
last method (ResultSet), 276, 280
lastXxxTime methods (BasicFileAttributes), 64
layer/ColorFrame.java, 545
Layers, 543–548
LayerUI class, 543–548
  extending, 544
  installUI, uninstallUI methods, 547
paint method, 544, 547
processXxxEvent methods, 545, 548
Layout algorithm, 647–648
Layout managers, using JavaBeans persistence for, 784
layoutPages method (Banner), 648
LCD displays, 621
LDAP (Lightweight Directory Access Protocol), 892
LD_LIBRARY_PATH environment variable, 996, 1033
Leaf icons, 442
Learning SQL (Beaulieu), 239
Leaves, 421, 428, 454
  icons for, 429–430, 442
Legacy data, converting into XML, 178
length method
  of Blob, 270
  of CharSequence, 8
  of Clob, 271
  of RandomAccessFile, 29, 33
LIB environment variable, 1033
LIKE operator (SQL), 243, 272
limit method (Buffer), 75
Line2D class, 553
lineTo method (shape classes), 559, 570
Linux operating system
  compiling invocation API, 1033
  Java deployment directory in, 875
  library path in, 996
  OpenSSL in, 872
  using GNU C compiler, 993
List boxes, 364–370
  constructing, 364
    from arrays/vectors, 376–377
    new components for each cell, 380
double-clicking in, 367
  fixed cell sizes in, 372
  layout orientation of, 365
  scrolling, 364, 372
  selecting multiple items in, 365, 367
  using with split panes, 515
values of
  computing when called, 371
  editing, 375–377
  rendering, 377–381
  very long, 371–372
List cell renderers, 377–381
List interface, 486
list method
  of Context, 964
  of Naming, 965
List model, 370–375
List selection events, 366
ListCellRenderer interface
  getListCellRendererComponent method, 378–381
  implementing, 378–379
ListDataListener interface, 371
list/ListFrame.java, 367
ListModel interface, 376
  getElementAt method, 371, 375
  getSize method, 371, 375
  implementing, 370–371
listRendering/FontCellRenderer.java, 380
ListResourceBundle class, 345
ListSelectionEvent class, 366
  getValueIsAdjusting method, 366
ListSelectionListener interface
  valueChanged method, 366, 370
ListSelectionModel interface
  selection modes, 394
  setSelectionMode method, 394, 406
LITTLE_ENDIAN constant (ByteOrder), 76
Little-endian order, 25, 70
Load time, 949
loadClass method
  of ClassLoader, 807, 809
  of URLClassLoader, 806
loadImage method (SimpleBeanInfo), 755–756
loadLibrary method (System), 994, 996
LOBs (large objects), 269–271
  creating empty, 271
  placing in database, 270
  reading, 269
  retrieving, 269–270
Local encoding schemes, 21
Local hosts, 192
Local names, 149
Local parameter and result objects, 972
Locale class, 307–311
  constructor, 308, 310
  debugging, 311
  forLanguageTag method, 310
  getCountry method, 311
  getDisplayCountry, getDisplayLanguage methods, 311
  getDisplayName method, 310–311, 313
  getInstance method, 329
  getLanguage method, 311
  get/setDefault methods, 309–310
toLanguageTag method, 311
toString method, 311
Locales, 306–311
and resources bundles, 342–343
  current, 337, 763
  default, 310
display names of, 310
  languages in, 307, 309
  locations in, 307
  no connection between character encodings and, 340
  numbers in, 313, 468
  predefined, 308
  supported by classes, 309
  variants in, 307, 343
lock method (FileChannel), 79–81
Locks, 79–81
  for the tail portion of a file, 80
  shared, 80
  unlocking, 80
@LogEntry annotation, 943
Logging
  code generation for, 919
  messages, adding to classes, 943
  RMI activity, 968–970
LoggingPermission class, 832
logging.properties file, 968
LoginContext class, 843
  constructor, 847
  getSubject method, 848
  login method, 843, 847
  logout method, 843, 848
LoginModule interface
  abort method, 858
  commit method, 858
  initialize method, 858
  login, logout methods, 858
Logins
  committed, 851
  modules for, 843–844
    custom, 849–858
  separating from action code, 851
LONG NVARCHAR data type (SQL), 301
long type
  printing, 14
  type code for, 44, 1011
  vs. C types, 997
  writing in binary format, 25
LONG VARCHAR data type (SQL), 301
LongBuffer class, 77
longList/LongListFrame.java, 373
longList/WordListModel.java, 374
Long-term storage, 779–780
lookingAt method (Matcher), 91
lookup method
  of Context, 964
  of Naming, 965
Lookup tables, 344
LookupOp class, 627–628
  constructor, 635
lookupPrintServices method (PrintServiceLookup), 660, 662
lookupStreamPrintServiceFactories method
  (StreamPrintServiceFactory), 664
LookupTable class, 628
lostOwnership method (ClipboardOwner), 674, 678
LSB (least significant byte), 25
LSOutput interface, 161
LSSerializer interface,
  &lt; entity reference (HTML, XML), 98
M
  Mac OS X operating system
    OpenSSL in, 872
    resources in, 342
  Mail headers, 230
  Mail messages, 230–233
  mail method (Desktop), 719
  mail/MailTest.java, 232
  mailto: URL prefix, 714
  main method
    executing, 804
      setting security managers in, 828
  makeShape method (ShapeMaker), 560–561
  makeVisible method (JTree), 434, 439
  Mandelbrot set, 622–623
  Mangling names, 993, 1010, 1013
  Manifest files, 34, 708, 731–732
  Map interface, 601
  map method (FileChannel), 69, 74
  Maps, using JavaBeans persistence for, 784
  mark method
    of Buffer, 78
    of InputStream, 4
  Marker annotations, 927
  markSupported method (InputStream), 4
  MarshalledObject class
    constructor, 986
    get method, 981, 986
  MaskFormatter class, 473–474
    constructor, 484
    get/setInvalidCharacters methods, 473, 485
    get/setPlaceholder methods, 485
    get/setPlaceholderCharacter methods, 474, 485
    get/setValidCharacters methods, 473, 484
    get/setValueContainsLiteralCharacters methods, 485
      overtype mode, 474
  Mastering Regular Expressions (Friedl), 86
  match attribute (XSLT), 174
  Matcher class
    end method, 86, 89, 92
    find method, 89, 92
    group, groupCount methods, 87, 92
    lookingAt method, 91
    matches method, 86
    quoteReplacement method, 92
    replaceAll, replaceFirst methods, 90, 92
    reset method, 92
    start method, 86, 89, 92
    matcher method (Pattern), 86, 91
    matches method (Matcher), 86
    match/HrefMatch.java, 89
  Matrices, transformations of, 585–586
  maxoccurs attribute (XML Schema), 124
  MAX_VALUE value, 80
  MD5 algorithm, 860–861
  MDI (multiple document interface), 524–525
  Memory addresses, vs. serial numbers, 40
  Memory mapping, 68–81
    modes of, 69
  memoryMap/MemoryMapTest.java, 71
  Message digests, 859–862
  Message signing, 862–865
  MessageDigest class
    digest method, 861–862
    extending, 860
    getInstance method, 860–862
    reset method, 862
    update method, 861–862
  MessageFormat class, 335–340, 347
    applyPattern method, 337
    constructor, 337
    format method, 337
    get/setLocale methods, 337
    ignoring the first limit, 339
  Messages
    formatting, 335–340
    placeholders in, 336–340
  Meta-annotations, 920, 933–935
  Metadata, 286–296
  Metal look-and-feel
    hot keys in, 467
    internal frames in, 524–527
    one-touch expand icons in, 515
    selecting multiple nodes in, 446
Method class implementing AnnotatedElement, 921
using JavaBeans persistence for, 784
Method verification errors, 820–821
Methods
adding logging messages to, 943
annotating, 920, 931
calling
  from native code, 1012–1019
  on another machine, 957
executing, 1018
getters/setters, generated automatically, 940
inherited from superclass, 743
instance, 1012–1016
names of
capitalization of, 742
mangling, 993, 1010, 1013
patterns for, 739–743
native, 989–1050
of annotation interfaces, 928
overriding, 932
signatures of
  encoding, 1010–1012
  generating from class files, 1012
static, 1016–1017
vetoable, 531
Metric system, 345
Microsoft
  compiler, 993–994
  invocation API in, 1033
  ODBC API of, 236
  SQL Server, 245
  single active statements in, 255
MIME (Multipurpose Internet Mail Extensions)
  reader or writer matching, 609
MIME types, 678–680
  for print services, 661
  MimeUtility class, 216
mioccurs attribute (XML Schema), 124
MissingResourceException, 343
Miter join, 572–573
Miter limit, 572
Mixed content, 97
  parsing, 117
Model-view-controller design pattern, 370
Modernist painting example, 161–162
Modified UTF-8 encoding, 26–28, 42, 340,
  999–1002
Mouse, double-clicking, 367
move method (Files), 61–62
moveColumn method (JTable), 399, 405
moveTo method (shape classes), 559–560, 570
moveToXXX methods (JInternalFrame), 542
moveToXXXRow methods (ResultSet), 278, 280
MSB (most significant byte), 25
Multicast lookup, of remote objects, 961
MULTILINE flag (Pattern), 86
Multiple-page printing, 646–649
MutableTreeNode interface
  implementing, 423
  setUserObject method, 423, 430
MySQL database, 245
N
  \n  as end-of-line character, 14, 104
  in e-mails, 230
  in regular expressions, 83–86
  NameAlreadyBoundException, 964
  NameCallback class, 850
    constructor, 857
    getDefaultName method, 857
    getPrompt method, 857
    get/setName methods, 857
  NameClassPair class, 962
    getName, getClassname methods, 965
  NamedNodeMap interface
    getLength method, 112
    item method, 112
  Namespaces, 146–149
    activating processing of, 124
    aliases (prefixes) for, 122, 148
    of attributes, 148
    of child elements, 147
    using class loaders as, 808
  Name/value pairs. See Properties
  Naming class
    bind, list, lookup, rebind, unbind methods, 965
  Naming patterns, 739–743
    add/remove/get, 743, 745
    for customizers, 771
    get/set, 741–744
    is/set, 742
    standard, 754
  NanoHTTPD web server, 967, 976, 983
  National character strings, 300
  National Institute of Standards and Technology, 186, 860
  native keyword, 990
Native applications
communicating with Java, 689–707
data transfers between Java and, 673
Native methods, 989–1050
accessing
array elements, 1019–1023
instance fields, 1005–1009
static fields, 1009–1010
strings, 999
analyzing exceptions, 1024
class references in, 1007
compiling, 993
enumerating keys with, 1038
error handling in, 1023–1028
implementing registry access functions in, 1036–1050
invoking Java constructors in, 1017–1018
linking to Java, 996
naming, 991–992
overloading, 991
reasons to use, 990
static, 991
native2ascii program, 341, 344, 347
NCHAR data type (SQL), 300–301
NCLOB data type (SQL), 301
Negative byte values, 475
Nervous text applet, 858
NetBeans builder environment, 725–802
    color property editor in, 758, 773
    composing beans in, 733–739
    importing beans into, 732
    opening customizer in, 772
    startup plugins for, 708
NetPermission class, 831
Networking, 185–233
    blocking worker threads indefinitely, 506
    debugging, 185
    getting huge images from, 611
    security risks of, 975
NetXxxArrayRegion functions (C), 1022
newBufferedXxx methods (Files), 61
NewByteArray function (C), 1037
NewDirectByteArray function (C), 1022
newDirectoryStream method (Files), 65, 67
newDocument method (DocumentBuilder), 159, 163, 179
newDocumentBuilder method
    (DocumentBuilderFactory), 100, 111, 160
newFactory method (RowSetProvider), 282, 285
newFileSystem method (FileSystems), 67–68
NewGlobalRef function (C), 1007
newInstance method of DocumentBuilderFactory, 100, 111
newInstance method of SAXParserFactory, 151, 154
newInstance method of TransformerFactory, 163
newInstance method of XMLInputFactory, 158
newInstance method of XLOutputFactory, 164, 171
newInstance method of XPathFactory, 141, 146
Newline character. See End-of-line character
NewObject function (C), 1019
newInstance method (Channels), 203
of Channels, 209
of Files, 61
newInstance method (XPathFactory), 146
newInstance method (SAXParserFactory), 151, 154
NewString function (C), 1002
NewStringUTF function (C), 1000–1001, 1003, 1037
newInstance method (TransformerFactory), 163, 182
next method
    of ResultSet, 252, 254, 274
    of XMLStreamReader, 158
nextElement method (Enumeration), 440, 1036, 1038
newPage method (CachedRowSet), 283, 285
NMTOKEN, NMTOKENS attribute types (DTDs), 117–118
Node interface
    appendChild method, 160, 163
    getAttributes method, 104, 112
    getChildNodes method, 101, 112
    getFirstChild method, 103, 112
    getLastChild method, 104, 112
    getLocalName method, 149
    getNamespaceURI method, 149
    getNextSibling method, 104, 112
    getNodeName method, 104, 112
    getNodeValue method, 104, 112
    getParentNode method, 112
    getPreviousSibling method, 112
    subinterfaces of, 101
Node renderer, 428
nodeChanged method (DefaultTreeModel), 433, 440
NodeList interface, 101
    getLength method, 101, 112
    item method, 101, 112, 120
Nodes (in trees), 421
    adding/removing, 433
    changes in, 434
child, 421, 424
collapsed, 435
connecting lines for, 426–427
currently selected, 431
editing, 435, 455
enumerating, 440–442
expanding, 435
handles for, 424, 428, 442
highlighting, 443
identifying, by tree paths, 431
making visible, 434
parent, 421, 424
rendering, 442–445
root, 421–429
row positions of, 433
selecting, 446
user objects for, 423
changing, 433
nodesChanged method (DefaultTreeModel), 440
Nondeterministic parsing, 117
Non-XML legacy data, converting into XML, 178
Normalization, 329–335
normalize method (Path), 58–59
Normalized color values, 620
Normalizer class, 330
normalize method, 333
NoSuchAlgorithmException, 862, 886
NoSuchElementException, 1039
NotBoundException, 965
notFilter method (RowFilter), 397, 407
NotSerializableException, 49
NT logins, 842
N-tier models, 238
NTLoginModule class, 844–845
NTUserPrincipal class, 845
NullPointerException, 1025
Number class
doubleValue method, 312
intValue method, 312
numberFilter method (RowFilter), 397, 407
NumberFormat class, 311–319, 472
format method, 313, 317
getAvailableLocales method, 309, 313, 317
currencyInstance method, 311, 317–318,
getNumberInstance method, 470
getPercentInstance method, 311, 317, 472
get/setMaximumXXXDigits methods, 317
get/setMinimumXXXDigits methods, 317
is/setGroupingUsed methods, 317
is/setParseIntegerOnly methods, 317
parse method, 312–313, 317
setCurrency method, 318
NumberFormatException, 464
numberFormat/NumberFormatTest.java, 313
Numbers
filtering, 397
floating-point, 306, 311, 472
formatting, 306, 311–319, 472
supported locales for, 313
with C, 997–999
in regular expressions, 85
printing, 14
reading
from files, 10
from ZIP archives, 11
using locales, 312
truly random, 883
writing in binary format, 25
NUMERIC data type (SQL), 245, 301
NVARCHAR data type (SQL), 301
O
Object class
close method, 37, 54
Object inspection tree, 453–462
Object references
passing by serializing, 971
transferring via clipboard, 689
Object serialization, 36–57
file format for, 42–48
MIME type for objects in, 679
modifying default, 48–50
of singletons, 50–52
serial numbers for, 38, 40
using for cloning, 54–57
vs. JavaBeans persistence, 779–780
with RMI, 971–974
ObjectInputStream class, 37
constructor, 41
readObject method, 37, 42, 49
ObjectOutputStream class, 37
constructor, 41
defaultWriteObject method, 49
writeObject method, 37, 41, 49
Objects
cloning, 54–57
communicating between, 956
compatibility with earlier versions of, 52–54
gradient, 581–582
setting, 551
solid, 581
Paint interface, 581
paintComponent method
  casting, 550
  of JComponent, 378–379
  of StrokePanel, 575
paintValue method (PropertyEditor), 762, 770
  implementing, 766
Paper margins, 639
Paper sizes, 345, 640
Parameter marshalling, 957–958
Parent nodes (in trees), 421, 424
parse method
  of DateFormat, 320, 326
  of DocumentBuilder, 111
  of NumberFormat, 312–313, 317
  of SAXParser, 151, 154
  of XMLReader, 183
Parsed character data, 116
ParseException, 312, 317
Parsers, 99–112
  checking uniqueness of IDs in, 118, 128
  pull, 156
  validating in, 113
Parsing, 99–112
  by URI class, 210
  nondeterministic, 117
  with XML Schema, 124
PasswordCallback class, 850
  constructor, 858
  getPrompt method, 858
  get/setPassword methods, 858
  isEchoOn method, 858
Password-protected resources, 214
Passwords, 850
Path class, 57, 59–60
  get, getFileName, getRoot methods, 59
  normalize, relativize methods, 58–59
  resolve, resolveSibling methods, 58–59
  toAbsolutePath method, 58–59
  toFile method, 59
pathFromAncestorEnumeration method, 441
Paths, 57–60
  absolute vs. relative, 9, 57–58
  resolving, 11, 58
  root component of, 57
  separators in, 9, 57
Paths class, 67
  get method, 58–59
Pattern class, 86
  compile method, 86, 91
  flags of, 86
  matcher method, 86, 91
  split method, 91
Patterns, 81–92
#PCDATA rule (DTDs), 116
PDF format, printing, 659
PEM (Privacy Enhanced Mail), 872
Pentium processor, little-endian order in, 25
Percentages, formatting, 311–319
Perl programming language, regular
  expressions in, 86
Permission class
  constructor, 842
  equals method, 834
  extending, 834
  getActions method, 834
  getName method, 837, 842
  hashCode method, 834
  implies method, 834, 836–836, 842
Permission files, 823
permission keyword, 829, 832
Permissions, 822–826
  call stack of, 825
  class hierarchy of, 824
  custom, 834–835
  for files, 829
  for users, 843
  implementing, 835–842
  implying other permissions, 836
  in policy files, 826–834
  mapping code sources to, 822
  order of, 834
  property, 833
  restricting to certain users, 845
  socket, 832
  targets of, 829–833
permissions/PermissionTest.java, 840
permissions/WordCheckPermission.java, 838
Permutations, of a string, 487–488
Persistence delegates, 784–788
  installing, 785
  persistenceDelegate/PersistenceDelegateTest.java, 789
persistentFrame/PersistentFrameTest.java, 782
Personal data, transferring, 880
Pixels
  affine transformations on, 627
  antialiasing, 601–604
  average value of, 629
  composing, 592–601
  interpolating, 581, 603–604, 627
  reading, 620
  setting individual, 619–626
PKCS #5 (Public Key Cryptography Standard), 882
Placeholder characters, 474
Placeholders (in messages), 336–340
PlainDocument class, 463, 471
Plugins, 873
  packaged as JAR files, 806
PNG image format, 608
  animated, 708
  printing, 659
Point class, using JavaBeans persistence for, 784
Point2D.Double class, 792
Pointer arithmetic, 802
Points (in typography), 640
Policy class, 823, 827
Policy files, 826–834, 975
  and Java Plug-in, 873
  building, 874–877
  debugging, 875
  locations for, 826
  parsing, 837
  platform-independent, 833
  referencing keystores in, 874
  system properties in, 833
  user roles in, 842, 851–858
  visibility of, 837
Policy managers, 826–834
PolicyTool program, 833
Polygons, 553, 559
populate method (CachedRowSet), 283, 285
Pop-up menus, for tray icons, 720
PopupMenu class, 720
Porter–Duff rules, 592–596
Portrait orientation, 640
Ports, 186–187
  in URIs, 211
position function (XPath), 177
position method (Buffer), 79
POSIX-compliant file systems, 64
PosixFileAttributes interface, 64
POST command, 224–226
@PostConstruct annotation, 932
PostConstruct interface, 931
PostgreSQL database, 245
  connections to, 249
  drivers for, 246
Postorder traversal, 441
postOrderEnumeration method
  (DefaultMutableTreeNode), 441, 444
post/PostTest.java, 228
PostScript format
  printing, 659, 664
  writing to, 664
Predefined character class names, 82–83, 85
@PreDestroy annotation, 932
PreDestroy interface, 931
premain method, 950
preOrderEnumeration method
  (DefaultMutableTreeNode), 441, 444
Prepared statements, 263–269
  caching, 264
  executing, 264
PreparedStatement interface
  clearParameters method, 269
  executeXXX methods, 264, 269
  setXXX methods, 263, 269
prepareStatement method (Connection), 263, 269,
  274–275, 279
  previous method (ResultSet), 276, 280
  previousPage method (CachedRowSet), 285
preVisitDirectory, postVisitDirectory methods
  of FileVisitor, 65
  of SimpleFileVisitor, 67
Primary keys, 273
Primitive types
  arrays of, 1022
  I/O in binary format, 5
    using JavaBeans persistence for, 784
Principal interface
  getName method, 849
Principals (logins), 844
Print dialog box, 637
  displaying page ranges in, 639, 647
  native, 638, 642
Print job attributes, 665
print method
  of Desktop, 714, 719
  of DocPrintJob, 662
  of JTable, 384, 386
  of Printable, 637, 645, 647, 649–650
  of PrinterJob, 637–639, 646
  of PrintWriter, 14–15, 1012–1016
  of String, 1013
Print preview, 649–659
Print request attributes, 665
Print service attributes, 665
Print services, 659–662
  document flavors for, 660–661
  for images, 660–662
  stream, 663–664
Printable interface
  implementing, 637, 642
  objects, printing, 659
  print method, 637, 645, 647, 649–650
printDialog method (PrinterJob), 637–638, 646
Printer graphics context, 648
PrinterException class, 637
defaultPage method, 646
defaultPage method, 646
getPrinterJob method, 637, 646
getPageDialog method, 640, 642, 646
print method, 637–639, 646
printDialog method, 637–638, 646
setPageable method, 647, 659
setPrintable method, 646
printf method
  in C, 997–999
  of PrintWriter, 14–15
Printf class, 1012–1016
printf1/Printf1.c, 998
printf1/Printf1.java, 998
printf1/Printf1Test.java, 999
printf2/Printf2.c, 1004
printf2/Printf2.java, 1003
printf2/Printf2Test.java, 1003
printf3/Printf3.c, 1015
printf3/Printf3.java, 1014
printf3/Printf3Test.java, 1014
printf4/Printf4.c, 1025
printf4/Printf4.java, 1027
printf4/Printf4Test.java, 1028
Printing, 636–672
  clipped areas, 639
  counting pages during, 639
  images, 637–647
  layout of, 647–648
  multiple-page, 646–649
  number of copies for, 665
  page orientation of, 587, 640
  paper sizes in, 640
  quality of, 668
  selecting settings for, 637
  starting, 637, 821
  text, 637–647
  using
    banding for, 639
    transformations for, 649
Printing attributes, 664–672
  adding, 667
  categories of, 666–667
  checking values of, 668
  hierarchy of, 666
  retrieving, 667
PrinterJob class (obsolete), 637
PrinterJobAttribute interface, 665
  printing attributes of, 668–671
PrinterJobAttributeSet interface, 665
PrintPreviewCanvas class, 649
PrintPreviewDialog class, 649–659
print/PrintComponent.java, 644
print/PrintTestFrame.java, 643
printQuality printing attribute, 668
PrintRequestAttribute interface, 665
  printing attributes of, 668–671
PrintRequestAttributeSet interface, 637, 665
PrintService interface
  createPrintJob method, 660, 662
  getAttributes method, 672
  getName method, 660
PrintServiceAttribute interface, 665
  printing attributes of, 668–671
PrintServiceAttributeSet interface, 665
PrintServiceLookup class
  lookupPrintServices method, 660, 662
printService/PrintServiceTest.java, 663
PrintStream class, 14
PrintWriter class, 10, 13–16
  checkError method, 14, 16
  constructor, 15
  print method, 14–15, 1012–1016
  printf method, 14–15
  println method, 14–15
Private keys, 862–874, 889–890
PrivilegedAction interface, 851
  implementing, 844
  run method, 844, 848
PrivilegedExceptionAction interface, 844
  run method, 848
process method (BeanInfoAnnotationProcessor), 938
processAnnotations method, 921–922
Processing instructions, 99
Processing tools, 919
processingEnv field (AbstractProcessor), 938
Processor interface, 938
processXxxEvent methods (LayerUI), 545, 548
Programs. See Applications

Progress bars, 501–505
  completion percentage in, 502
  indeterminate, 502
  minimum/maximum values of, 502
  on a splash screen, 708–710
Progress monitors, 505–508
  cancellation requests in, 506
  for input streams, 509–514
  timers in, 506
progressBar/ProgressBarFrame.java, 502
ProgressMonitor class, 505–508
  close method, 505, 514
  constructor, 513
  isCanceled method, 506, 514
  setMillisToDecideToPopup, setMillisToPopup methods, 506
  setNote method, 514
  setProgress method, 505, 514
ProgressMonitorInputStream class, 509–514
  constructor, 514
  read method, 509
progressMonitorInputStream/TextFrame.java, 510
progressMonitor/ProgressMonitorFrame.java, 506

Properties
  Boolean-valued, 158
  bound, 745–746
  changing in property inspector, 738, 743
  constrained, 531–543, 746–754
  constructing objects from, 785–787
  editing values in customizer, 773
  in NetBeans, 737
  indexed, 744–745
  inherited, hiding, 737, 754
  names of, 739–743
    decapitalized, 742
  read-only, read/write, write-only, 741
  recovering names of, 937–938
  saving and restoring, 791
  simple, 744
  transient, 788–792
  types of, 743–754
  vs. instance fields, 728
Properties class, 94
@Property annotation, 936–937
Property editors, 737, 758–770
  adding to GUI, 773
  for colors, 758
  GUI-based, 765–770
  string-based, 762–765
  vs. customizers, 770
writing, 762–770

Property files, 94
  character encoding of, 344
  event handlers in, 902
  for resources bundles, 342–343
  for string resources, 342
  for strings, 343–344
  no passwords in, 231
  reading, 231

Property inspectors, 726, 765
  changing properties in, 738, 743
  displaying current properties in, 766–767
  listing bean property names, 737
  suppressing visibility of properties in, 737

Property permissions, 833

PropertyChange event, 737, 745

PropertyChangeEvent class, 753, 772–773
  constructor, 752
  getNewValue method, 543, 746, 752
  getOldValue method, 746, 752
  getPropertyName method, 543, 746, 752

PropertyChangeListener interface
  propertyChange method, 746, 751

PropertyChangeSupport class, 745
  add/removePropertyChangeListener methods, 751
  constructor, 751
  fireIndexedPropertyChange method, 751
  firePropertyChange method, 751
  getPropertyChangeListeners method, 751

PropertyDescriptor class, 754, 758, 937
  constructor, 757, 761
  getPropertyType, getReadMethod, getWriteMethod methods, 757
  get/setValue method, 769
  setAsText method, 762–763, 766, 769

PropertyEditor interface
  getAsText method, 762–763, 766, 769
  getCustomEditor method, 762, 767, 770
  getJavaInitializationString method, 763–764, 770
  get/setValue method, 769
  getTags method, 762–763, 770
  implementing, 762
  isPaintable method, 762, 766, 770
  paintValue method, 762, 766, 770
  setAsText method, 762–763, 770
  supportsCustomEditor method, 762, 767, 770

PropertyEditorSupport class, 763
  extending, 762

PropertyPermission class, 830

PropertyVetoException, 527–532, 746–747, 752
  constructor, 543, 753
getPropertyChangeEvent method, 753
Protection domains, 824
ProtectionDomain class
    constructor, 826
getCodeSource method, 826
    implies method, 826
Prototype cell values, 372
Proxy class, using JavaBeans persistence for, 784
Proxy objects, 921, 955
    communicating, 956
    for annotated methods, 922
    for annotation interfaces, 928
Public certificates, keystore for, 874
PUBLIC identifier (DTDs), 161
Public key ciphers, 862–870, 888–892
    speed of, 889
Pull parsers, 156
PushbackInputStream class, 10
    constructor, 12
    unread method, 12
put method
    of Bindings, 897
    of ByteBuffer, 75
    of CharBuffer, 76
    of ScriptEngine, 897
    of ScriptEngineManager, 897
putClientProperty method (JComponent), 426, 431, 534
putNextEntry method (ZipOutputStream), 34–35
putXxx methods (ByteBuffer), 70, 76

R
    \r
        as end-of-line character, 14, 104
        in e-mails, 230
        in regular expressions, 83
raiseSalary method (Employee), 1005–1007
Random class, 883
Random-access files, 28–33
RandomAccessFile class, 28–33, 69
    constructor, 32
    getChannel method, 74
    getChannel method, 74
    getFilePointer method, 28, 33
    length method, 29, 33
    seek method, 28, 33
randomAccess/RandomAccessTest.java, 31
Randomness, 883
Raster class
    getDataElements method, 621–622, 625
    getPixel, getPixels methods, 620, 625
rasterImage/RasterImageFrame.java, 623
read method
    of CipherInputStream, 888
    of FileInputStream, 2
    of ImageIO, 608, 616
    of ImageReader, 617
    of InputStream, 2–3
    of ProgressMonitorInputStream, 509
    of Readable, 7–8
    of Reader, 5
    of ZipInputStream, 33
Readable interface, 6
    read method, 7–8
ReadableByteChannel interface, 202
readAllXxx methods (Files), 60–61
readAttributes method (Files), 64
Reader class, 2, 5, 7
    read method, 5
READER data source (print services), 661
readExternal method (Externalizable), 50
readFixedString method (DataInput), 29–30
read/fontdialog.xml, 135
readFully method (DataInput), 27
read/gridbag.dtd, 137
read/GridBagPane.java, 131
read/GridBagTest.java, 129
read/gridbag.xsd, 138
readline method (BufferedReader), 16
readObject method
    of Date, 49
    of ObjectInputStream, 37, 42, 49
    of XMLDecoder, 781, 801
ReadOnlyBufferException, 69
readResolve method, 51
readThumbnail method (ImageReader), 618
readUTF method (DataInput), 26–27
readXxx methods (DataInput), 26–27, 37
REAL data type (SQL), 245, 301
rebind method
  of Context, 964
  of Naming, 965
Rectangle class, using JavaBeans persistence for, 784, 786
Rectangle2D class, 553
RectangularShape class, 554
Redundancy elimination, 781
ref attribute (XML Schema), 123
Reflection
  accessing fields of another class with, 821
  constructing
class tree with, 447
  static field names with, 325
  enumerating fields from a variable with, 455
  for bean properties, 741, 754
ReflectPermission class, 831
regedit program, 1034
regexFilter method (RowFilter), 397, 407
regex/RegexTest.java, 87
register method (Activatable), 983, 986
Registry editor, 1034, 1039
Registry keys, 1036–1050
Regular expressions, 81–92
  boundary matchers in, 83
  character classes in, 83
  characters in, 83
  escapes in, 17, 82, 84
  filtering, 397
  formatters for, 476
  grouping in, 82, 84, 87
  in DTDs, 116
  predefined character classes in, 82–83, 85
  quantifiers in, 82, 84
  replacing all matches of, 90
  set operations in, 84
relative method (ResultSet), 276, 280
Relative URLs, 874
Relativization, of an absolute URL, 212
relativize method (Path), 58–59
Relax NG (Regular Language for XML Next Generation), 114
releaseSavepoint method (Connection), 298–299
ReleaseStringChars function (C), 1002
ReleaseStringUTFChars function (C), 1001–1003
ReleaseXxxArrayElements functions (C), 1020, 1023
ReleaseXxxArrayRegion functions (C), 1023
reload method (DefaultTreeModel), 434, 440
remaining method (Buffer), 78–79
Remote interface, 959, 971
Remote methods, 956–958
  logging, 968–970
  parameters of, 970–980
  reliability of, 959
  return values of, 970–980
Remote objects, 957
  activating, 980–987
  comparing, 980
  interfaces for, 959
  MIME type for, 679
  passing, 970
  registering, 961
  transferring, 971
Remote references
  cost of, 971
  no clone method for, 980
  transferring, 971
  with multiple interfaces, 979
RemoteException, 959–960, 979–980
remove method
  of AttributeSet, 672
  of DocumentFilter, 484
  of SystemTray, 723
removeActionListener method (TrayIcon), 724
removeCellEditorListener method (CellEditor), 420
removeColumn method (JTable), 398, 405
removeElement method (DefaultListModel), 377
removeNodeFromParent method (DefaultTreeModel), 433, 439
removePropertyChangeListener method
  implementing, 745
  of Component class, 746, 753
  of Customizer, 772
  of PropertyChangeSupport class, 751
removeTabAt method (JTabbedPane), 518, 523
removeTreeModelListener method (TreeModel), 454, 462
removeUpdate method (DocumentListener), 464, 467
removeVetoableChangeListener method
  implementing, 747
  of JComponent, 753
  of VetoableChangeSupport, 752
RendererImage interface, 659
Rendering
  cells, 408–420
columns, 390–391
headers, 409
lists, 377–381
nodes, 442–445
Rendering hints, 550–551, 601–608
Rendering pipeline, 550–552
RenderingHints class, 601
constructor, 608
renderQuality/RenderQualityTestFrame.java, 605
replaceAll method (DocumentFilter), 470–471, 484
replaceFirst, replaceAll methods (Matcher), 90, 92
required keyword, 844
#REQUIRED attribute default (DTDs), 118
requisite keyword, 844
RescaleOp class, 627
constructor, 635
Rescaling operation, 627
reset method
of Buffer, 78
of InputStream, 4
of Matcher, 92
of MessageDigest, 862
reshape method (JInternalFrame), 526, 542
resolve method (Path), 58–59
resolveEntity method (EntityResolver), 115, 121
resolveSibling method (Path), 58–59
Resolving
classes, 804
relative URLs, 212
@Resource annotation, 302, 932
Resource bundles, 341–346
loading, 344
locating, 342–343
lookup tables for, 344
naming, 344
searching for, 344
Resource editor, 342
Resource interface, 931
ResourceBundle class
extending, 344–345
getBundle method, 342–344, 346
getKeys method, 346
getObject method, 346
getString method, 344, 346
getStringArray method, 346
handleGetObject method, 346
ResourceBundle.Control class
getCandidateLocales method, 343
Resources
annotations for managing, 932
hierarchy of, 343
Resources interface, 931
Response headers, 214, 216
Response page, 223
Result interface, 178, 301
Result sets (SQL)
accessing columns in, 253
analyzing, 252
closing, 255
for multiple queries, 255
iterating over rows in, 274
metadata for, 287
numbering rows in, 276
order of rows in, 253
retrieving multiple, 272–273
scrollable, 274–277
updatable, 274, 277–281
ResultSet interface, 281
absolute method, 276, 280
cancelRowUpdates method, 278, 281
close method, 255
concurrency values, 275, 277, 279, 281
deleteRow method, 278, 281
findColumn method, 255
first, last, beforeFirst, afterLast methods, 276, 280
getBlob, getClob methods, 269–270
getConcurrency method, 276–277, 280
getMetaData method, 287, 296
getRow method, 276, 280
type method, 276, 279
getWarnings method, 258
getXXX methods, 253–254
insertRow method, 278, 280
isClosed method, 255
isFirst, isLast, isBeforeFirst, isAfterLast methods, 277, 280
iteration protocol, 252
moveToXXXRow methods, 278, 280
next method, 252, 254, 274
previous method, 276, 280
relative method, 276, 280
type values, 275, 277, 279, 281
updateRow method, 278, 281
updateXXX methods, 277–278, 281
ResultSetMetaData interface, 287
ggetColumnXXX methods, 287, 296
@Retention annotation, 920, 933
Retention interface, 931
Retention policies, 933
Retirement calculator example, 346–362
INDEX

1102

INDEX
Index

Runtime class
  exit method, 821–842
runtimeAnnotations/ActionListenerFor.java, 925
runtimeAnnotations/ActionListenerInstaller.java, 923
RuntimePermission class, 830
S, \s, \S, in regular expressions, 83
S (short), type code, 44, 1011
Sample values, 620
Sandbox, 822–826, 879
  no needed, for corporate code, 873
SASL (Simple Authentication and Security Layer), 892
Save points, 297–298
Savepoint interface
  getSavepointXxx methods, 299
SAX (Simple API for XML) parser, 99, 150–155
  activating namespace processing in, 152
SAXParseException
  getLineNumber, getColumnNumber methods, 121
SAXParser class
  parse method, 151, 154
SAXParserFactory class
  is/setNamespaceAware methods, 154
  isValidating, setValidating methods, 154
  newInstance method, 151, 154
  newSAXParser method, 151, 154
  setFeature method, 152
SAXResult class, 178
sax/SAXTest.java, 153
SAXSource class, 177–178
  constructor, 183
Scalar functions, 271–272
scale method (Graphics2D), 584, 589
Scaling, 584–585
  and interpolating pixels, 603–604
Scanner class, 16, 202
schema element (XML Schema), 124
Schemas (in databases), 295
Script engines, 894–895
  adding variable bindings to, 896
  implementing Java interface, 899
  invoking functions in, 899–900
ScriptContext interface, 897
  get/setErrorWriter methods, 898
  get/setReader methods, 898
  get/setWriter methods, 898
ScriptEngine interface
  createBindings method, 897
eval method, 895–897
get method, 897
getContext method, 898
put method, 897
ScriptEngineFactory interface
  getExtensions method, 895
  getMethodCallSyntax method, 899
  getMimeTypes method, 895
  getNames method, 895
ScriptEngineManager class
  get method, 897
  getEngineByXxx, getEngineFactories methods, 894–895
  put method, 897
Scripting languages, 893–906
  advantages of, 894
  supported, 894
Scripts
  accessing classes through Java interfaces in, 900
  compiling, 901
  executing, 901
  concurrently in multiple threads, 896
  invoking, 894–895
  multiple, 896
  redirecting I/O of, 898
  server-side, 222–230
  using Java method call syntax in, 899
script/ScriptTest.java, 903
Scroll pane
  with lists, 364, 372
  with tabbed panes, 518–519
  with tables, 383
  with trees, 434–435
scrollPathToVisible method (JTree), 434, 439
Secret key, generating, 883
SecretKey interface, 883
SecretKeySpec class, 887
Secure random generator, 883
SecureRandom class, 883
  setSeed method, 883
Securing Java (McGraw/Felten), 825
Security, 802–892
  bytecode verification, 815–821
  class loaders, 804–815
  code signing, 873–880
  different levels of, 859
  digital signatures, 858–873
  encryption, 880–892
  user authentication, 842–858
Security managers, 821–842
  implementing, 822
  in RMI applications, 975
reading policy files, 975
Security policy, 822
  in activation groups, 982
SecurityException, 822, 824
SecurityManager class, 975
checkExit method, 821, 823–824
checkPermission method, 823–825, 835–836
checkRead method, 825
SecurityPermission class, 831
seek method (RandomAccessFile), 28, 33
“Seek forward only” mode (ImageInputStream), 610
select attribute (XSLT), 175
SELECT statement (SQL), 242–243
  executing, 252
  for LOBs, 269
  multiple, in a query, 272
  not supported in batch updates, 298
Selection model, 394
separator constant, 9
Separators, in paths, 9, 57
sequence element (XML Schema), 124
Serial numbers, 38, 40
  vs. memory addresses, 40
SerialCloneable class, 54
serialClone/SerialCloneTest.java, 55
@Serializable annotation, 934
Serializable interface, 37, 44, 934, 971
SerializablePermission class, 831
Serialization, 36–57
  file format for, 42–48
  MIME type for objects in, 679
  modifying default, 48–50
  of singletons, 50–52
  serial numbers for, 38, 40
  using for cloning, 54–57
  vs. JavaBeans persistence, 779–780
  with RMI, 971–974
serialTransfer/SerialTransferFrame.java, 686
serialver program, 52
serialVersionUID constant, 52
server/EchoServer.java, 195
Servers
  accessing, 209–230
  calls to, 966
  connecting clients to, 185–193
  implementing, 193–202
  in distributed programming, 953–956
  installing proxies on, 955
  invoking programs, 222
Server-side scripts, 222–230
  sending comma-separated files, 226
ServerSocket class, 193–202
  accept method, 194, 197
  close method, 197
  constructor, 197
Service provider interfaces, 609
ServiceCenter interface, 979
  getReturnAuthorization method, 979
SERVICE_FORMATTED data source (print services), 661
Servlets, 222, 913–918
  hot deployment of, 808
Set interface
    containsAll, equals methods, 837
Set operations, 84
setAllowsChildren method
  (DefaultMutableTreeNode), 428, 431
setAllowUserInteraction method (URLConnection),
  212, 214, 221
setAsksAllowsChildren method
  (DefaultMutableTreeNode), 430–431
setAsText method (PropertyEditor), 762–763, 770
setAttribute, setAttributeNS methods (Element),
  160, 163
setAutoCreateRowSorter method (JTable), 384, 386, 395
setAutoResizeMode method (JTable), 393, 404
setBinaryStream method (Blob), 270
setBottomComponent method (JSplitPane), 517
setCalendar method (DateFormat), 327
setCellEditor method (TableColumn), 411, 419
setCellRenderer method
  of JList, 378, 381
  of TableColumn, 419
setCellSelectionEnabled method (JTable), 394, 405
setCharacterStream method (Clob), 271
setClip method (Graphics), 589–591, 639
setClosable method (JInternalFrame), 541
setClosed method (JInternalFrame), 531, 542
setClosedIcon method (DefaultTreeCellRenderer),
  445
setColumnSelectionAllowed method (JTable), 394, 405
setCommand method (RowSet), 283, 285
setComparator method (DefaultRowSorter), 396, 406
setComponentAt method (JTabbedPane), 523
setComposite method (Graphics2D), 551, 594, 601
setConnectTimeout method (URLConnection), 212, 221
setContentHandler method (XMLReader), 183
setContentPane method (JInternalFrame), 542
setContents method (Clipboard), 674, 677
setContextClassLoader method (Thread), 807, 815
setContinuousLayout method (JSplitPane), 515, 517
setCrc method (ZipEntry), 36
setCurrency method (NumberFormat), 724
setDecomposition method (Collator), 335
setDefault method (Locale), 309–310
setDefaultNamespace method (XMLStreamWriter), 172
setDefaultRenderer method (JTable), 409
setDisplayTableName method (FeatureDescriptor), 757
setDoXXX methods (URLConnection), 212–213, 220, 225
setDragEnabled method (Swing), 691, 696
setDragMode method (JDesktopPane), 534, 541
setDropAction method (TransferSupport), 706
setDropMode method (Swing), 700, 705
setEditable method (JEditorPane), 496
setEditor method (JSpinner), 492
setEntityResolver method (DocumentBuilder), 115, 121
setErrorHandler method (DocumentBuilder), 120–121
setErrorListener method (Encoder), 801
setErrorWriter method (Swing), 700, 705
setError method (JEditorPane), 496
setErrorMethod (Swing), 691, 696
setExpert method (FeatureDescriptor), 757
setFeature method (SAXParserFactory), 152
setFillViewportHeight method (JTable), 384, 386
setFixedCellXXX methods (JList), 375
setFocusLostBehavior method (JFormattedTextField), 469, 483
setFixedCell method (JInternalFrame), 525, 542
setFrom method (JavaMail), 231
setGroupingUsed method (NumberFormat), 317
setHeaderXXX methods (TableColumn), 409, 419
setHidden method (FeatureDescriptor), 757
setIcon method (JInternalFrame), 542
setIconAt method (JTabbedPane), 523
setIconifiable method (JInternalFrame), 541
setIfModifiedSince method (URLConnection), 212, 214, 221
setIgnoringElementContentWhitespace method (DocumentBuilderFactory), 119, 122
setImage, setImageAutoSize methods (TrayIcon), 724
setImageURL method (SplashScreen), 713
setIndeterminate method (JProgressBar), 502, 513
setInput method (ImageReader), 617
setInvalidCharacters method (MaskFormatter), 473, 485
setItem method (JLayer), 445
setMinimum method (JInternalFrame), 541
setMaximum method of JInternalFrame, 528, 542
of JProgressBar, 502, 513
setMaxLength method (TableColumn), 392, 406
setMethod method of ZipEntry, 35
of ZipOutputStream, 35
setMillisToDecideToPopup, setMillisToPopup methods (ProgressMonitor), 506
setMinimum method (JProgressBar), 502, 513
setMinimumXXXDigits, setMaximumXXXDigits methods (NumberFormat), 317
setMinWidth method (TableColumn), 392, 406
setMnemonicAt method (JTabbedPane), 518, 524
setName method of FeatureDescriptor, 757
of NameCallback, 857
setNameSpaceAware method of DocumentBuilderFactory, 124, 149, 152, 160
of SAXParserFactory, 154
setNext method (ProgressMonitor), 514
setNumberFormat method (DateFormat), 327
setObject method (Customizer), 772–773, 779
SetObjectArrayElement function (C), 1020, 1023–1024
setOneTouchExpandable method (JSplitPane), 515, 517
setOpenIcon method (DefaultTreeCellRenderer), 445
setOutput method (ImageWriter), 618
setOutputProperty method (Transformer), 164
setOverwriteMode method (DefaultFormatter), 483
setPage method (JEditorPane), 496, 500
setPageable method (PrinterJob), 647, 659
setPageSize method (CachedRowSet), 283, 285
setPaint method (Graphics2D), 551, 581, 583
setParseIntegerOnly method (NumberFormat), 317
setPassword method
  of PasswordCallback, 858
  of RowSet, 283–284
setPerspectiveDelegate method (Encoder), 801
setPixel, setPixels methods (WritableRaster), 620, 626
setPlaceholder method (MaskFormatter), 485
setPlaceholderCharacter method (MaskFormatter), 474, 485
setUp method (TrayIcon), 724
setPreferredHeight method (JComponent), 466
setPreferredWidth method (TableColumn), 392, 406
setPrefix method (XMLStreamWriter), 172
setPrintable method (PrinterJob), 646
setProgress method (ProgressMonitor), 505, 514
setProperty method
  of System, 975
  of XMLInputFactory, 156, 158
setPropertyEditorClass method
  of Descriptor, 937
  of PropertyDescriptor, 758, 761
setPrototypeCellValue method (List), 375
setReader method (ScriptContext), 898
setReadTimeout method (URLConnection), 212, 221
setRenderingHint, setRenderingHints methods
  (Graphics2D), 551, 601–604, 607
setRequestProperty method (URLConnection), 212, 214, 221
setResizable method
  of JInternalFrame, 541
  of TableColumn, 393, 406
setRightComponent method (JSplitPane), 517
setRootVisible method (JTree), 428, 430
setRowHeight method (JTable), 393–394, 404
setRowMargin method (JTable), 394, 404
setRowSelectionAllowed method (JTable), 394, 405
setRowSorter method (JTable), 395, 405
Sets, comparing, 837
setSavepoint method (Connection), 299
setSecurityManager method (System), 828, 975
setSeed method (SecureRandom), 883
setSelected method (JInternalFrame), 527, 542
setSelectedIndex method (JTabbedPane), 518, 523
setSelectionMode method
  of JList, 365, 369
  of ListSelectionModel, 394, 406
set/SetTest.java, 949
setShortDescription method (FeatureDescriptor), 757
setShowsRootHandles method (JTree), 428, 430
setSize method (ZipEntry), 36
setSortable method (DefaultRowSorter), 395, 406
setSoTimeout method (Socket), 190–191
SetStaticXxxField functions (C), 1009–1010
setStrength method (Collator), 335
setText, setStringPainted methods
  (JProgressbar), 502, 513
setTextConverter method (TableRowSorter), 406
setStroke method (Graphics2D), 551, 572, 580
setSubject method (JavaMail), 231
setTabComponentAt method (JTabbedPane), 519, 524
setTabLayoutPolicy method (JTabbedPane), 518, 523
setTableName method (CachedRowSet), 284–285
setText method
  of JavaMail, 231
  of JFormattedTextField, 469
setTimeZone method (DateFormat), 327
setTitle method (event listeners), 774
setTitleAt method (JTabbedPane), 523
setToolTip method (TrayIcon), 724
setTopComponent method (JSplitPane), 517
setToXxx methods (AffineTransform), 587, 589
setTransferHandler method
  of JComponent, 692, 695
  of JFrame, 705
setTransform method (Graphics2D), 587, 589
setURL method (RowSet), 283–284
setUseCaches method (URLConnection), 212, 214, 221
setUsername method (RowSet), 283–284
setUserObject method (MutableTreeNode), 423, 430
setValidating method
  of DocumentBuilderFactory, 119, 122
  of SAXParserFactory, 154
setValidCharacters method (MaskFormatter), 473, 484
setValue method
  of AbstractSpinnerModel, 487, 494
  of JFormattedTextField, 468–469, 482
  of JProgressbar, 502, 513
  of JSpinner, 492
  of PropertyEditor, 769
setValueAt method (TableModel), 390, 414
setValueContainsLiteralCharacters method (MaskFormatter), 485
setVisible method (JInternalFrame), 526, 542
setVisibleRowCount method (JList), 365, 369
setWidth method (TableColumn), 393, 406
setWriter method (ScriptContext), 898
setXxx methods (PreparedStatement), 263, 269
SetXxxArrayRegion functions (C), 1022
SetXxxField functions (C), 1006, 1010, 1038
SGML (Standard Generalized Markup Language), 95
SHA-1 (Secure Hash Algorithm), 43, 859–861
Shape interface, 553, 571
ShapeMaker class
  getPointCount method, 560
  makeShape method, 560–561
ShapePanel class, 561
Shapes, 552–570
  antialiasing, 602
  append method, 560, 570
  clipping, 550, 589–592
  closePath method, 559, 570
  combining, 570–572
  control points of, 560–561
  creating, 551
  curveTo, lineTo, quadTo methods, 559, 570
  drawing, 550–553
  filling, 550–551, 581
  moveTo method, 559–560, 570
  transformations of, 550
shape/ShapeTest.java, 561
Shared libraries, 996, 1033
  initialization blocks in, 996
Shear, 584–585
shear method (Graphics2D), 584, 589
short type
  printing, 14
  type code for, 44, 1011
  vs. C types, 997
  writing in binary format, 25
ShortBuffer class, 77
ShortLookupTable class, 628
  constructor, 636
shouldSelectCell method (CellEditor), 413, 420
show method (JInternalFrame), 542
showInternalXxxxDialog methods (JOptionPane), 533
shutdownXxx methods (Socket), 202
Side files, 935
Signatures, 858–873
  generating, 1012
  mangling, 1010–1012
signedBy keyword, 875
signed/ReadApplet.java, 877
Simple properties, 744
Simple types (XML Schema), 122
SimpleBeanInfo class, 754
  loadImage method, 755–756
SimpleDateFormat class
  toPattern method, 493
SimpleDoc class, 660
  constructor, 662
SimpleFileVisitor class, 66
  postVisitDirectory method, 67
  preVisitDirectory method, 67
  visitFile, visitFailed methods, 66–67
SimpleJavaFileObject class
  extending, 909
  getCharContent method, 909, 913
  openOutputStream method, 909, 913
SimpleLoginModule class, 849
  checkLogin method, 849
  initialize method, 850
SimpleScriptContext class, 897
  simpleType element (XML Schema), 123
SimulatedActivity class, 502
Single value annotations, 927
Singletons
  of the splash screen, 709
  serializing, 50–52
SINGLE_TREE_SELECTION mode (TreeSelectionModel), 446
SISC Scheme scripting engine, 894–895, 902–903
size method
  of BasicFileAttributes, 64
  of Files, 63–64
skip method (InputStream), 3
skipBytes method (DataInput), 27
SMALLINT data type (SQL), 245, 301
SMTP (Simple Mail Transport Protocol), 230–233
SOAP (Simple Object Access Protocol), 956
Socket class
  connect method, 191
  constructor, 190–191
  getInputStream method, 189–190, 194
  getOutputStream method, 190, 194
  isClosed, isConnected methods, 191
  isXxxShutdown, shutdownXxx methods, 202
  setSoTimeout method, 190–191
Socket permissions, 832
SocketChannel class, 202
  open method, 202, 209
SocketPermission class, 829
Sockets, 189
  half-closing, 201–202
Sockets (cont.)
  interrupting, 202–209
  opening, 189, 821
  timeout, 190–191
socket/SocketTest.java, 189
SocketTimeoutException, 191, 201
Software developer certificates, 878–880
Software updates, 719
Solaris operating system
  compiling invocation API, 1033
  using Sun compiler, 993
sort method (Collections), 329
Source files
  annotations, 940
  character encoding, 340–341
  location of, 908
  reading from disk, 908
Source interface, 177, 301
sourceAnnotations/BeanInfoAnnotationProcessor.java, 940
sourceAnnotations/Property.java, 936
Source-level annotations, 935–943
Space, in e-mail URLs, 224, 714
SPARC processor, big-endian order in, 25
SpinnerModel class, 493
SpinnerListModel class, 486, 493
SpinnerNumberModel class, 486, 492–493
spinner/PermutationSpinnerModel.java, 490
Spinners, 485–494
  increment value in, 485
  traversal order in, 486
spinner/SpinnerFrame.java, 488
Splash screens, 708–713
  drawing on, 708–710
  replacing with follow-up window, 710
SplashScreen class
  close method, 713
  createGraphics method, 713
  getBounds method, 713
  getImageURL method, 713
  getSplitScreen method, 709, 713
  update method, 709, 713
splashScreen/SplashScreenTest.java, 710
split method
  of Pattern, 91
  of String, 17
Split panes, 514–517
  orientation of, 515
  splitter bar of, 515
splitPane/SplitPaneFrame.java, 516
sprintf function (C), 1002
SQL (Structured Query Language), 239–244
  changing data inside databases, 244
  commands in, 247
  data types in, 245, 300–301
  equality testing in, 243
  escapes in, 271–272
  exceptions in, 256–258
  executing statements in, 251–262
  interactive tutorial for, 244
  keywords in, 242
  reading instructions from a file, 259
  strings in, 243
  vs. Java, 265
  warnings in, 256
  wildcards in, 243
SQLException, 256–258, 276
  and rolling back, 297
  and save points, 299
  getError Code, getNextException, getSQLState, iterator methods, 256–257
SQLPermission class, 832
SQLWarning, 256, 276
  getNextWarning method, 258
SQLXML data type (SQL), 301
Square cap, 572–573
SQuirreL SQL client, 287
SRC, SRC_Xxx composition rules, 594–595
sRGB standard, 621
SSL (Secure Sockets Layer), 892
Standard extensions, 805
StandardJavaFileManager interface, 908–909
  getJavaFileObjectsFromXxx methods, 912
start method (Matcher), 86, 89, 92
startDocument method (ContentHandler), 154
startElement method (ContentHandler), 150–155
startDateEnumeration function (C), 1038–1039
Statement interface, 251–262
  addBatch method, 298–300
  close, closeOnCompletion methods, 254–255
  constructor, 802
  execute method, 254, 259, 273–274
  executeBatch method, 298, 300
  executeQuery method, 252–253, 276
  executeUpdate method, 252, 254, 274, 297
  getMoreResults method, 273
  getResultSet method, 254
  getUpdateCount method, 254, 273
  getWarnings method, 258
  isClosed method, 254
  using for multiple queries, 255
Statements (SQL)
    closing, 255
    concurrently open, 255
    executing, 251–262
    grouping into transactions, 296–301
    in batch updates, 298
    multiple, 255
    prepared, 263–269
    truncations in, 257
Static fields, accessing from native code, 1009–1010
Static initialization blocks, 994
Static methods, calling from native code, 1016–1017
StAX parser, 156–159, 164–172
    namespace processing in, 156
    no indented output in, 165
StAXSource class, 177
stax/StAXTest.java, 157
stopCellEditing method (CellEditor), 413–414, 420
Stored procedures, 271–272
Stream filters
    combining, 9–12
    nesting, 9–12
Streaming parsers, 99, 149–159
StreamPrintService class, 664
StreamPrintServiceFactory class, 664
    getPrintService method, 664
    lookupStreamPrintServiceFactories method, 664
StreamResult class, 178
    constructor, 164
Streams, 1–12
    and Unicode, 2
    buffered, 10–12
    byte-oriented, 2
    chaining, 10
    encrypted, 887–888
    filters for, 509
    for print services, 663–664
    hierarchy of, 4–9
    input, 2, 100, 509–514
    output, 2
    raw byte, adding capabilities to, 10
    writing objects to, 36–57
StreamSource class, 177
    constructor, 182
    transform method, 178
String class, 8
    compareTo method, 328
    print method, 1013
    split method, 17
    trim method, 103, 312
STRING data source (print services), 661
String parameters, 999–1005
String-based property editors, 762–765
StringBuilder class, 8, 77
StringBuilder class, 8, 29, 470, 510
Strings
    accessing from native code, 999–1005
    encoding, 21, 340
    in Unicode, 305
    fixed-size, I/O of, 29–30
    in SQL, 243
    internationalizing, 343
    ordering, 328
    patterns for, 81–92
    permutations of, 487–488
    printing, 14
    property editors working with, 762–765
    sorting, 329
    translating, 341
    using JavaBeans persistence for, 784
    writing in binary format, 25
StringSelection class, 674, 680
stringToValue method (AbstractFormatter), 475, 483
Stroke interface, 572
StrokePanel class, 575
Strokes, 550, 572–581
    antialiasing, 602
    dash patterns of, 574
    end cap styles of, 572–574
    join styles of, 572–573
    setting, 551
    thickness of, 572
stroke/StrokeTest.java, 575
Stubs, 957–958
StyledDocument interface, 463
StylePad demo, 463
Subject class, 848
    doAs, doAsPrivileged methods, 844–845, 848
Subjects (logins), 844
subSequence method (CharSequence), 9
subtract method (Area), 571–572
Subtrees, 424
    adding nodes to, 434
    collapsed and expanded, 426
Suetonius, Gaius Tranquillus, 810
    sufficient keyword, 844
Sun compiler, 993
SunJCE ciphers, 881, 887
@SupportedAnnotationTypes annotation, 938
SupportedValuesAttribute interface, 665
supportsBatchUpdates method (DatabaseMetaData), 298, 300
supportsCustomEditor method (PropertyEditor), 762, 770
overriding, 767
supportsResultSetXxx methods (DatabaseMetaData), 276, 281
@SuppressWarnings annotation, 932
SuppressWarnings interface, 931
SVG (Scalable Vector Graphics), 161–162
Swing, 363–548
compatibility of, with future releases, 780
data transfer in, 691–707
generating dynamic code for, 913
loading time in, 708
using JavaBeans persistence for, 784
Swing components
attaching verifiers to, 471
desktop panes, 524–527
drag-and-drop behavior of, 696–698
editor pane, 494–501
internal frames, 524–527
layers, 543–548
layout managers for, 125–139
lists, 364–370
organizing, 514–548
progress bars, 501–505
progress monitors, 505–508
spinners, 485–494
split panes, 514–517			tabbed panes, 518–524
tables, 381–420
text, 462–501
trees, 420–462
Swing Connection newsletter, 476
SwingWorker class, 502
Symmetric ciphers, 881–882
speed of, 889
SyncProviderException, 284–285
System class
loadLibrary method, 994, 996
setProperty method, 975
setSecurityManager method, 828, 975
System class loader, 805
SYSTEM identifier (DTDs), 114, 161
System properties, in policy files, 833
System tray, 719–724
System.in class, 14
System.out class, 14
SystemTray class, 719–724
add method, 720, 723
getSystemTray method, 720, 723
getTrayIconSize method, 723
isSupported method, 719, 723
remove method, 723
systemTray/SystemTrayTest.java, 721
T
t escape (SQL), 271
\t, in regular expressions, 83
Tabbed panes, 518–524
adding/removing tabs in, 518
labels of, 518
scrolling, 518–519
tabbedPane/TabbedPaneFrame.java, 520
Table cell renderers, 391, 408
Table index values, 395
Table model, 382, 386–390
updater after cells were edited, 414
TableCellEditor class
getTableCellEditorComponent method, 411, 413, 420
implementing, 411, 413
tableCellRender/ColorTableCellEditor.java, 417
tableCellRender/ColorTableCellRenderer.java, 417
TableCellRenderer interface
getTableCellRendererComponent method, 408, 419
implementing, 408
tableCellRender/PlanetTableModel.java, 415
tableCellRender/TableCellRendererFrame.java, 414
TableModel class, 392–393, 398
constructor, 406
setCellEditor method, 411, 419
setCellRenderer method, 419
setHeaderXxx methods, 409, 419
setResizable, setWidth methods, 393, 406
setWidth methods, 392, 406
TableColumnModel interface, 392
getColumn method, 405
TableModel interface, 395
getColumnClass method, 390, 404
getColumnClassName method, 388, 390
get/setValueAt methods, 386–387, 390, 414
getXxxCount methods, 386, 388, 390
implementing, 386
isCellEditable method, 390, 410
tableModel/InvestmentTableModel.java, 388
tableRowColumn/PlanetTableFrame.java, 399
executing scripts in, 896
for new pages in JEditorPane, 496
Internet connections with, 197–201
referencing class loaders, 806–808
Three-tier model, 238–239
throw function (C), 1023–1024, 1028
Throwable class, 1023
throwNew function (C), 1023–1024, 1028
thumbnails, 611
Tiling windows, 529
time
  daylight saving, 327
editing, 472
  formatting, 319–328
  literals for, 271
TIME data type (SQL), 245, 271, 301
time of day service, 186
time picker, 487
time zones, 319
timeouts, 190–191
timer class, 506
TIMESTAMP data type (SQL), 245, 271, 301
timeZone Class
  getAvailableIDs, getDisplay, getDisplayName,
  getID, getTimeZone methods, 327
  isDaylightTime, useDaylightTime methods, 328
titlePositionEditor class, 759
toAbsolutePath method (Path), 58–59
toArray method (AttributeSet), 60
Tödter, Kai, 728
toFile method (Path), 59
toLowerCase method (Locale), 311
tool interface
  run method, 911
toolkit class
  getSystemClipboard method, 674, 677
toolProvider class
  getSystemJavaCompiler method, 907
tools.jar file, 907
tooltips, for tray icons, 720
toPath method (File), 60
toPattern method (SimpleDateFormat), 493
top-level windows, opening, 821
toString method
  of Annotation, 930
  of ByteBuffer, 24–25
  of CharSequence, 9
  of Currency, 319
  of DefaultFormatter, 473
  of Locale, 311
  of TableStringConverter, 396, 406
of variable, 456
transactions, 296–301
  committing, 296–297
  rolling back, 296–297
transfer handlers, 692–693
  adding, 696
transfer wrapper, 686
Transferable interface, 678–680
  getTransferData method, 678
  getTransferDataFlavors method, 680
  implementing, 674, 680, 689
  isDataFlavorSupported method, 678
TransferHandler class, 696–698
  canImport method, 699, 705
  constructor, 696
  createTransferable method, 696, 705
  exportAsDrag method, 697–698
  exportDone method, 698
  getTransferData method, 678
  getTransferDataFlavors method, 680
  implement transfer method, 674, 680
  isDataFlavorSupported method, 678
TransferHandler.DropLocation class, 701
  getDropPoint method, 706
TransferSupport class
  getComponent, getDataFlavors,
    getDropActions, getUserDropAction
  methods, 706
  getDropLocation method, 700–701, 706
  get/setDropAction method, 706
  getTransferable method, 700
  isDrop method, 701, 706
transferText/TextTransferFrame.java, 675
transform method
  of Graphics2D, 551, 587, 589
  of StreamSource, 178
  of Transformer, 164, 177
transformations, 550, 583–589
  affine, 586, 627
  composing, 584, 586
  fundamental types of, 584–585
  matrices for, 585–586
  order of, 585
  setting, 551
  using for printing, 649
Transformer class
  setOutputProperty method, 164
  transform method, 164, 177
TransformerFactory class
  newInstance method, 163
  newTransformer method, 163, 182
transform/makehtml.xsl, 176
transform/makeprop.xsl, 177
transform/TransformTest.java, 179
transient keyword, 48–49
Transient properties, 788–792
Transitional events, 366
translate method (Graphics2D), 584, 589, 648
Translation, 584–585
Transparency, 592–601
Traversal order
   in a spinner, 486
   in a tree
      breadth-first vs. depth-first, 440
      postorder, 441
Tray icons, 719–720
   pop-up menus for, 720
   tooltips for, 720
TrayIcon class, 720
   add/removeActionListener methods, 724
   constructor, 724
   displayMessage method, 724
   get/setImage methods, 724
   get/setPopupMenu methods, 724
   get/setTooltip methods, 724
   is/setImageAutoSize methods, 724
Tree events, 445–453
Tree model
   constructing, 422, 454
   custom, 453–462
   default, 422
Tree parsers, 99
Tree paths, 431–440
   constructing, 434, 441
Tree selection listener, 445
   adding to a tree, 445
TreeNode interface, 430, 436
   children, getChildXxx methods, 439
   getAllowsChildren method, 430
   getParent method, 439, 441
   isLeaf method, 428, 430
TreePath class, 432–433
   getLastPathComponent method, 432–433, 439
TreeSelectionEvent class
   getPath method, 453
   getPaths method, 447, 453
TreeSelectionListener interface
   implementing, 445–453
   valueChanged method, 445, 447, 452
TreeSelectionModel interface, 446
tree/SimpleTreeFrame.java, 425
tree/TreeModel.java, 438
   trim method (String), 103, 312
True Odds: How Risks Affect Your Everyday Life (Walsh), 859
tryLock method (FileChannel), 79–81
try-with-resources statement, 7, 65
   for database connections, 255
   with locks, 80
ts escape (SQL), 271
Type codes, 44, 1011
Type conversions, unchecked, 802
Type definitions (XML Schema), 122
  anonymous, 124
  nesting, 123
TYPE_BYTE_GRAY image type, 623
TYPE_INT_ARGB image type, 620–621
Typesafe enumerations, 50–52
U
u, in masks, 474
\u, in regular expressions, 83
UDP (User Datagram Protocol), 190
UIManager class, 408
unbind method
  of Context, 964
  of Naming, 965
Unchecked type conversions, 802
UnicastRemoteObject class, 960, 981
  exportObject method, 960
  extending, 968
Unicode encoding
  and byte-oriented streams, 2
  character order in, 328
  converting to binary data, 13, 20
  in property files, 344
  in regular expressions, 85
  normalization forms in, 330
  using for all strings, 305
UNICODE_CASE flag (Pattern), 86
uninstallUI method (LayerUI), 547
Units of measurement, 98
UNIX family of operating systems
  authentications in, 842–843
  end-of-line character in, 14
  file names starting with a period in, 827
  Java deployment directory in, 875
  paths in, 57
UNIX_LINES flag (Pattern), 86
UnixLoginModule class, 844
UnixNumericGroupPrincipal class, 844
UnixPrincipal class, 843–844
  getName method, 843
UnknownHostException, 991
Unparsed external entity, 119
unread method (PushbackInputStream), 12
UnsatisfiedLinkError, 991
UnsupportedFlavorException, 678
UNWRAP_MODE mode (Cipher), 881
update method
  of Cipher, 881, 884, 886, 888
  of MessageDigest, 861–862
  of SplashScreen, 709, 713
UPDATE statement (SQL), 244, 277
  executing, 252, 254, 269
  in batch updates, 298
  truncations in, 257
  vs. methods of ResultSet, 278
updateRow method (ResultSet), 278, 281
updateXxx methods (ResultSet), 277–278, 281
Upper case, turning characters to, 471
URI class, 210
  getAuthority method, 211
  getFragment method, 211
  getHost method, 211
  getPath method, 211
  getPort method, 211
  getQuery method, 211
  getScheme method, 211
  getSchemeSpecificPart method, 211
  getUserInfo method, 211
  nomailto support in, 714
URLs (Uniform Resource Identifiers), 210
  absolute vs. relative, 211
  base, 212
  for e-mails, 714
  hierarchical, 211
  namespace, 146–149
  opaque vs. nonopaque, 211
  schemes for, 211
URI Syntax class, 668
URL class, 210–212
  openConnection method, 212, 220
  openStream method, 220
  possible schemes for, 210
URL class
  openStream method, 210
URL data source (print services), 661
URLClassLoader class, 805–806
  constructor, 815
  loadClass method, 806
URLConnection class, 210, 212–222
  connect method, 212, 214, 221
  getContent method, 213, 222
  getContentXxx methods, 213, 216, 222
  getDate method, 213, 216, 222
  getExpiration method, 213, 216, 222
  getHeaderField method, 213, 222
  getHeaderFieldKey method, 213–215, 221
URLConnection class (cont.)
  getHeaderFields method, 213, 215, 221
  getInputStream method, 213, 222, 225, 227
  getLastModified method, 213, 216, 222
  getOutputStream method, 213, 222, 225
  get/setAllowUserInteraction methods, 212, 214, 221
  get/setConnectTimeout methods, 212, 221
  get/setDoInput methods, 212–213, 220
  get/setDoOutput methods, 212–213, 220
  get/setIfModifiedSince methods, 212, 214, 221
  get/setReadTimeout methods, 212, 221
  get/setRequestProperty methods, 212, 214, 221
  get/setUseCaches methods, 212, 214, 221
  getUseCaches method, 221
  setDoOutput method, 225
urlConnection/URLConnectionTest.java, 217
URLDecoder class
  decode method, 230
UREncoder class
  decode method, 229
URLs (Uniform Resource Locators), 210
  attaching parameters to, 224
  codebase, 968
  connections via, 210
  constructing from strings, 473
  encoding, 224–225
  for databases, 246
  for local files, 828
  for namespace identifiers, 147
  relative vs. absolute, 874
  for DTDs, 114
  resolving, 212
  RMI, 961
URNs (Uniform Resource Names), 210
US Letter paper, 640
U.S. Metric Association, 345
US-ASCII encoding, 22
useDaylightTime method (TimeZone), 328
User coordinates, 584
User names, 850
User objects, 423
  changing, 433
Users
  authentication of, 842–858
  role-based, 842–858
  permissions for, 845
  preferences of, 79
UTF-16 encoding, 13, 26, 999
UTF-8 encoding, 26, 999
  modified, 26–28, 42, 340, 999–1002
V
  V (void), type code, 1011
  Validation, 112–139
    activating, 119
    adding to classes, 48
    of input, 464, 467–485
  valueChanged method
    of ListSelectionListener, 366, 370
    of TreeSelectionListener, 445, 447, 452
  valueForPathChanged method (TreeModel), 455, 462
  value-of element (XSLT), 175
  valueToString method (AbstractFormatter), 475, 483
  VARCHAR data type (SQL), 245, 301
  Variable class, 455
    toString method, 456
  Variables
    annotating, 920, 930
    binding, 896
    fields of, 455
    scope of, 897
  Variants, in locales, 307, 343
  Vendor name, of a reader, 609
  Verifiers, 471–472, 815–821
  verifier/VerifierTest.java, 817
  verify method (InputVerifier), 471
  VeriSign, Inc., 867, 869, 873
  Version number, of a reader, 609
  Versioning, 52–54
  VERTICAL, VERTICAL_WRAP values (JList), 365
  VERTICAL_SPLIT value (JSplitPane), 515
  Vetoable change listeners, 531
  VetoableChangeSupport class, 753
  VetoableChangeListener interface, 746–747
    implementing, 531
    vetoableChange method, 532, 543, 752
    VetoableChangeSupport class, 747–748
      addVetoableChangeListener method, 752
      constructor, 752
      fireVetoableChange method, 747, 753
      getVetoableChangeListeners method, 753
      removeVetoableChangeListener method, 752
  Vetoing, 531–543
view/ViewDB.java, 288
visitFile, visitFileFailed methods
  of FileVisitor, 65
  of SimpleFileVisitor, 66–67
Visual Basic programming language, 726–802
  controls in, 726
  bean properties in, 727, 741–742
Visual representation, 238
W

\w, \W, in regular expressions, 83
walkFileTree method (Files), 65, 67
Warehouse interface, 959, 972, 979
warehouse1/WarehouseClient.java, 963
warehouse1/WarehouseImpl.java, 960
warehouse1/Warehouse.java, 959
warehouse1/WarehouseServer.java, 962
warehouse2/Book.java, 977
warehouse2/Product.java, 973
warehouse2/WarehouseImpl.java, 977
warehouse2/Warehouse.java, 972
warehouse2/WarehouseServer.java, 978
WarehouseImpl class, 960–961, 963, 968, 979, 981
warning method (ErrorHandler), 120–121
Warnings
  in applets, 837
  in SQL, 256
  suppressing, 932
WBMP image format, 608
WeakReference object, 457
Web applications, 955
  connection management in, 301–303
Web containers, 932
Web crawlers, 151
  with SAX parser, 152
  with StAX parser, 157
Web pages, dynamic, 913–918
Web services (WS-*), 956
Web Start applications, 239
WebRowSet interface, 282
WHERE clause (SQL), 243
Whitespace, 103
  ignoring, while parsing, 103
  in regular expressions, 85
  in XML Schema, 125
Wilde, Oscar, 306
Win32RegKey class, 1036, 1039
Win32RegKey_NameEnumeration class, 1038–1039
win32reg/Win32RegKey.c, 1041
win32reg/Win32RegKey.java, 1039
win32reg/Win32RegKeyTest.java, 1049
Windows look-and-feel
cascading/tiling in, 527
trees in, 426–427
Windows operating system
activating telnet in, 186
authentication in, 845
classpath in, 247
compiling invocation API, 1033
cut and paste in, 672–673
drag and drop in, 691
dynamic linking in, 1030
end-of-line character in, 14
glob syntax in, 65
Java deployment directory in, 875–876
multiple document interface of, 524–525
paths in, 9, 57
permissions in, 832
resources in, 342
system tray in, 719
using Microsoft compiler, 993–994
Windows registry, 1034–1050
  accessing from native code, 1036–1050
  keys in, 1036
Windows-1250 encoding, 22
WordCheckPermission, 836
Working directory, 9
wrap method (ByteBuffer), 24, 76, 78
WRAP_MODE mode (Cipher), 881
WritableByteChannel interface, 202
WritableRaster class, 620
  setDataElements method, 622, 626
  setPixel, setPixels methods, 620, 626
write method
  of CipherOutputStream, 888
  of Files, 60–61
  of ImageIO, 608, 616
  of ImageWriter, 611, 618
  of OutputStream, 2, 4
  of Writer, 5
  writeAttribute, writeCharacters, writeEmptyElement,
    writeEndDocument, writeEndElement methods
      (XMLStreamWriter), 165, 172
writeData, writeComment, writeDTD methods
      (XMLStreamWriter), 172
writeExternal method (Externalizable), 50
writeFixedString method (DataIO), 29–30
writeInsert method (ImageWriter), 612, 619
writeObject method
  of Date, 49
  of ObjectOutputStream, 37, 41, 49
  of XML encoder, 801
Writer class, 2, 5, 7
  write method, 5
write/RectangleComponent.java, 168
writeStartDocument method (XMLStreamWriter), 164,
  172
writeStartElement method (XMLStreamWriter), 165,
  172
writeStatement method (XMLEncoder), 801
writeType methods (DataOutput), 25
writeUTF method (DataOutput), 26, 28
write/XMLWriteFrame.java, 166
write/XMLWriteTest.java, 165
writeXXX methods (DataOutput), 28, 37
WSDL (Web Services Description Language), 956

X
\x, in regular expressions, 83
X Windows System, cut and paste in, 672–673
X.509 format, 866
XHTML (Extensible HyperText Markup Language), 151
XML (Extensible Markup Language), 93–184
advantages of, 956
annotated version of the standard, 96
case sensitivity of, 96
case sensitivity of, 96
for JavaBeans persistence, 780
hierarchical structures in, 94–95
in databases, 301
Java support of, 805
namespaces in, 146–149
vs. HTML, 96
XML documents
building without namespaces, 159–160
DTDs in, 96, 114
format of, 95
generating, 159–172
from non-XML legacy data, 178
HTML files from, 173–176
plain text from, 176–179
locating information in, 139–146
parsing, 99–112
structure of, 96, 100, 113
validating, 112–139
writing, 161
with StAX, 164–172
XML Schema, 113–114, 122–125
attributes in, 124
parsing with, 124
referencing in XML documents, 122
repeated elements in, 124
type definitions in, 122
anonymous, 124
nesting, 123
whitespace in, 125
XMLDecoder class
constructor, 801
get/setExceptionListener methods, 801
readObject method, 781, 801
XMLEncoder class, 780, 784
constructor, 801
writeObject, writeStatement methods, 801
XMLInputFactory class
createXMLStreamReader method, 158
newInstance method, 158
setProperty method, 156, 158
xmlns attribute (XML), 148
XMLOutputFactory class
createXMLStreamWriter method, 164, 171
newInstance method, 164, 171
XMLReader interface
implementing, 178
parse method, 183
setContentHandler method, 183
XMLStreamReader interface
close method, 172
setDefaultNamespace, setPrefix methods, 172
writeAttribute, writeCharacters, writeEmptyElement, writeEndDocument, writeEndElement methods, 165, 172
writeCDATA, writeComment, writeDTD methods, 172
writeStartDocument method, 164, 172
writeStartElement method, 165, 172
XMLStreamWriter interface, 164
close method, 172
setDefaultNamespace, setPrefix methods, 172
writeAttribute, writeCharacters, writeEmptyElement, writeEndDocument, writeEndElement methods, 165, 172
writeCDATA, writeComment, writeDTD methods, 172
writeStartDocument method, 164, 172
writeStartElement method, 165, 172
XMLWriter class, 785
XOR composition rule, 594
XPath (XML Path Language), 139–146
elements/attributes in, 140
expressions in, 141
XPath interface
evaluate method, 141, 146
XPathFactory class
newInstance method, 141, 146
newPath method, 146
xpath/XPathTester.java, 142
xs: prefix, 122
xsd: prefix, 122
xsd:attribute element (XML Schema), 124
xsd:choice element (XML Schema), 124
xsd:complexType element (XML Schema), 123
dxsd:element element (XML Schema), 123
xsd:enumeration element (XML Schema), 123
xsd:schema element (XML Schema), 124
xsd:sequence element (XML Schema), 124
xsd:simpleType element (XML Schema), 123
xsi namespace alias, 122
xsl:apply-templates element (XSLT), 174
xsl:output element (XSLT), 174
xsl:template element (XSLT), 174
xsl:value-of element (XSLT), 175
XSLT (Extensible Stylesheet Language Transformations), 161, 172–184
  copying attribute values in, 175
  stylesheets in, 172–184
  templates in, 174
XSLT processor, 173

Z
  \z, \Z, in regular expressions, 84
Z (boolean), type code, 44, 1011
ZIP archives, 33–36
  reading, 33
    numbers from, 11
  writing, 34
ZIP file systems, 79–81
ZipEntry class
  constructor, 35
  getName method, 35
  get/setCrc methods, 35–36
  get/setSize methods, 35–36
  isDirectory method, 35
  setMethod method, 35
ZipFile class
  constructor, 36
  entries, getEntry, getInputStream, getName methods, 36
ZipInputStream class, 5, 33
  closeEntry, getNextEntry methods, 33–34
  constructor, 34
  read method, 33
ZipOutputStream class, 5, 34
  closeEntry, putNextEntry methods, 34–35
  constructor, 35
  setLevel, setMethod methods, 35