ORACLE PRESS

Java™

CORE JAVA

Volume II: Advanced Features

TWELFTH EDITION

ORACLE

Cay S. Horstmann

FREE SAMPLE CHAPTER

Core Java

Volume II: Advanced Features

Twelfth Edition

This page intentionally left blank

Core Java

Volume II: Advanced Features

Twelfth Edition

Cay S. Horstmann

Pearson

Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Cover image: Leyland/Shutterstock
Figure 11.51: Shao.Chun Wang/123RF
Figure 9.2: Screenshot © Eclipse Foundation
Figure 10.4: Screenshot © 1999 . 2022 HHD Software Ltd
Figures 4.1, 4.2, 4.3, 4.4, 4.5, 5.3, 5.4, and 12.4: Screenshot © Microsoft 2022
Figures 3.3, 4.9, and 8.2: Screenshot © Mozilla Foundation
Figure 4.7: Screenshot © 2022 USPS
Figures 4.6, 5.6, 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 10.3, 10.9, 11.1, 11.2, 11.3, 1.4, 11.5, 11.7, 11.8, 11.9, 11.10, 11.14, 11.15, 11.16, 11.17, 11.18, 11.19, 11.20, 11.21, 11.22, 11.23, 11.24, 11.25, 11.26, 11.28, 11.29, 11.30, 11.39, 11.44, 11.47, 11.54, 11.55, 11.56, 11.60, and 11.62: Screenshot © 2022 Oracle

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The views expressed in this book are those of the author and do not necessarily reflect the views of Oracle.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the United States, please contact international@pearsoned.com.

Visit us on the Web: informit.com

Library of Congress Preassigned Control Number: 2022930917

Copyright © 2022 Pearson Education Inc.

Portions copyright © 1996-2013 Oracle and/or its affiliates. All Rights Reserved.

Oracle America Inc. does not make any representations or warranties as to the accuracy, adequacy or completeness of any information contained in this work, and is not responsible for any errors or omissions.

Microsoft and/or its respective suppliers make no representations about the suitability of the information contained in the documents and related graphics published as part of the services for any purpose. All such documents and related graphics are provided "as is" without warranty of any kind. Microsoft and/or its respective suppliers hereby disclaim all warranties and conditions with regard to this information, including all warranties and conditions of merchantability, whether express, implied or statutory, fitness for a particular purpose, title and non-infringement. In no event shall Microsoft and/or its respective suppliers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of information available from the services. The documents and related graphics contained herein could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Microsoft and/or its respective suppliers may make improvements and/or changes in the product(s) and/or the program(s) described herein at any time. Partial screen shots may be viewed in full within the software version specified.

Microsoft® Windows®, and Microsoft Office® are registered trademarks of the Microsoft Corporation in the U.S.A. and other countries. This book is not sponsored or endorsed by or affiliated with the Microsoft Corporation.

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearson.com/permissions/.

ISBN-13: 978-0-13-787107-0
ISBN-10: 0-13-787107-4

ScoutAutomatedPrintCode

Pearson's Commitment to Diversity, Equity, and Inclusion

Pearson is dedicated to creating bias-free content that reflects the diversity of all learners. We embrace the many dimensions of diversity, including but not limited to race, ethnicity, gender, socioeconomic status, ability, age, sexual orientation, and religious or political beliefs.

Education is a powerful force for equity and change in our world. It has the potential to deliver opportunities that improve lives and enable economic mobility. As we work with authors to create content for every product and service, we acknowledge our responsibility to demonstrate inclusivity and incorporate diverse scholarship so that everyone can achieve their potential through learning. As the world's leading learning company, we have a duty to help drive change and live up to our purpose to help more people create a better life for themselves and to create a better world.

Our ambition is to purposefully contribute to a world where:

- Everyone has an equitable and lifelong opportunity to succeed through learning.
- Our educational products and services are inclusive and represent the rich diversity of learners.
- Our educational content accurately reflects the histories and experiences of the learners we serve.
- Our educational content prompts deeper discussions with learners and motivates them to expand their own learning (and worldview).

While we work hard to present unbiased content, we want to hear from you about any concerns or needs with this Pearson product so that we can investigate and address them.

- Please contact us with concerns about any potential bias at https://www.pearson.com/report-bias.html.

This page intentionally left blank

Contents

Preface ... *xvii*

Acknowledgments ... *xxi*

Chapter 1: Streams ... 1

1.1 From Iterating to Stream Operations ... 2
1.2 Stream Creation ... 5
1.3 The `filter`, `map`, and `flatMap` Methods ... 11
1.4 Extracting Substreams and Combining Streams ... 13
1.5 Other Stream Transformations ... 15
1.6 Simple Reductions ... 16
1.7 The Optional Type ... 17
1.7.1 Getting an Optional Value ... 18
1.7.2 Consuming an Optional Value ... 18
1.7.3 Pipelining Optional Values ... 19
1.7.4 How Not to Work with Optional Values ... 20
1.7.5 Creating Optional Values ... 21
1.7.6 Composing Optional Value Functions with `flatMap` ... 22
1.7.7 Turning an Optional into a Stream ... 23
1.8 Collecting Results ... 26
1.9 Collecting into Maps ... 31
1.10 Grouping and Partitioning ... 35
1.11 Downstream Collectors ... 36
1.12 Reduction Operations ... 41
1.13 Primitive Type Streams ... 43
1.14 Parallel Streams ... 49

Chapter 2: Input and Output ... 55

2.1 Input/Output Streams ... 56
2.1.1 Reading and Writing Bytes ... 56
2.1.2 The Complete Stream Zoo ... 59

2.1.3 Combining Input/Output Stream Filters 64
2.1.4 Text Input and Output 68
2.1.5 How to Write Text Output 68
2.1.6 How to Read Text Input 70
2.1.7 Saving Objects in Text Format 72
2.1.8 Character Encodings 75
2.2 Reading and Writing Binary Data 78
2.2.1 The `DataInput` and `DataOutput` Interfaces 78
2.2.2 Random-Access Files 81
2.2.3 ZIP Archives 85
2.3 Object Input/Output Streams and Serialization 89
2.3.1 Saving and Loading Serializable Objects 89
2.3.2 Understanding the Object Serialization File Format 94
2.3.3 Modifying the Default Serialization Mechanism 101
2.3.4 The `readResolve` and `writeReplace` Methods 104
2.3.5 Versioning 107
2.3.6 Using Serialization for Cloning 110
2.3.7 Deserialization and Security 113
2.4 Working with Files 115
2.4.1 Paths 115
2.4.2 Reading and Writing Files 118
2.4.3 Creating Files and Directories 119
2.4.4 Copying, Moving, and Deleting Files 120
2.4.5 Getting File Information 123
2.4.6 Visiting Directory Entries 124
2.4.7 Using Directory Streams 126
2.4.8 ZIP File Systems 130
2.5 Memory-Mapped Files 131
2.5.1 Memory-Mapped File Performance 131
2.5.2 The Buffer Data Structure 139
2.6 File Locking 141
2.7 Regular Expressions 143
2.7.1 The Regular Expression Syntax 144
2.7.2 Matching an Entire String 149
2.7.3 Finding All Matches in a String 149

2.7.4 Groups 151
2.7.5 Splitting along Delimiters 154
2.7.6 Replacing Matches 154
2.7.7 Flags 155

Chapter 3: XML 159

3.1 Introducing XML 160
3.2 The Structure of an XML Document 162
3.3 Parsing an XML Document 165
3.4 Validating XML Documents 175
3.4.1 Document Type Definitions 176
3.4.2 XML Schema 185
3.4.3 A Practical Example 188
3.5 Locating Information with XPath 194
3.6 Using Namespaces 199
3.7 Streaming Parsers 203
3.7.1 Using the SAX Parser 203
3.7.2 Using the StAX Parser 208
3.8 Generating XML Documents 212
3.8.1 Documents without Namespaces 213
3.8.2 Documents with Namespaces 213
3.8.3 Writing Documents 214
3.8.4 Writing an XML Document with StAX 217
3.8.5 An Example: Generating an SVG File 222
3.9 XSL Transformations 223

Chapter 4: Networking 235

4.1 Connecting to a Server 235
4.1.1 Using Telnet 235
4.1.2 Connecting to a Server with Java 238
4.1.3 Socket Timeouts 240
4.1.4 Internet Addresses 242
4.2 Implementing Servers 244
4.2.1 Server Sockets 244
4.2.2 Serving Multiple Clients 247
4.2.3 Half-Close 251

4.2.4 Interruptible Sockets 252
4.3 Getting Web Data 259
4.3.1 URLs and URIs 259
4.3.2 Using a `URLConnection` to Retrieve Information 262
4.3.3 Posting Form Data 269
4.4 The HTTP Client 279
4.4.1 The `HttpClient` Class 279
4.4.2 The `HttpRequest` class and Body Publishers 279
4.4.3 The `HttpResponse` Interface and Body Handlers 280
4.4.4 Asynchronous Processing 281
4.5 Sending E-Mail 287

Chapter 5: Database Programming 291

5.1 The Design of JDBC 292
5.1.1 JDBC Driver Types 293
5.1.2 Typical Uses of JDBC 294
5.2 The Structured Query Language 295
5.3 JDBC Configuration 301
5.3.1 Database URLs 302
5.3.2 Driver JAR Files 302
5.3.3 Starting the Database 303
5.3.4 Registering the Driver Class 304
5.3.5 Connecting to the Database 304
5.4 Working with JDBC Statements 307
5.4.1 Executing SQL Statements 308
5.4.2 Managing Connections, Statements, and Result Sets 311
5.4.3 Analyzing SQL Exceptions 312
5.4.4 Populating a Database 315
5.5 Query Execution 319
5.5.1 Prepared Statements 319
5.5.2 Reading and Writing LOBs 326
5.5.3 SQL Escapes 328
5.5.4 Multiple Results 329
5.5.5 Retrieving Autogenerated Keys 330
5.6 Scrollable and Updatable Result Sets 331
5.6.1 Scrollable Result Sets 331

5.6.2 Updatable Result Sets 334
5.7 Row Sets 338
5.7.1 Constructing Row Sets 339
5.7.2 Cached Row Sets 339
5.8 Metadata 343
5.9 Transactions 353
5.9.1 Programming Transactions with JDBC 353
5.9.2 Save Points 354
5.9.3 Batch Updates 354
5.9.4 Advanced SQL Types 357
5.10 Connection Management in Web and Enterprise Applications .. 358

Chapter 6: The Date and Time API 361

6.1 The Time Line 362
6.2 Local Dates 366
6.3 Date Adjusters 372
6.4 Local Time 373
6.5 Zoned Time 375
6.6 Formatting and Parsing 379
6.7 Interoperating with Legacy Code 384

Chapter 7: Internationalization 387

7.1 Locales 388
7.1.1 Why Locales? 388
7.1.2 Specifying Locales 389
7.1.3 The Default Locale 392
7.1.4 Display Names 392
7.2 Number Formats 395
7.2.1 Formatting Numeric Values 395
7.2.2 The `DecimalFormat` Class 399
7.2.3 Currencies 402
7.3 Date and Time 403
7.4 Collation and Normalization 407
7.5 Message Formatting 413
7.5.1 Formatting Numbers and Dates 413
7.5.2 Choice Formats 416
7.6 Text Input and Output 418

7.6.1 Text Files ... 418
7.6.2 Line Endings ... 418
7.6.3 The Console ... 419
7.6.4 Log Files ... 420
7.6.5 The UTF-8 Byte Order Mark ... 420
7.6.6 Character Encoding of Source Files ... 420
7.7 Resource Bundles ... 421
7.7.1 Locating Resource Bundles ... 422
7.7.2 Property Files ... 423
7.7.3 Bundle Classes ... 424
7.8 A Complete Example ... 426

Chapter 8: Scripting, Compiling, and Annotation Processing ... 443

8.1 Scripting for the Java Platform ... 444
8.1.1 Getting a Scripting Engine ... 444
8.1.2 Script Evaluation and Bindings ... 445
8.1.3 Redirecting Input and Output ... 447
8.1.4 Calling Scripting Functions and Methods ... 448
8.1.5 Compiling a Script ... 450
8.1.6 An Example: Scripting GUI Events ... 451
8.2 The Compiler API ... 456
8.2.1 Invoking the Compiler ... 457
8.2.2 Launching a Compilation Task ... 457
8.2.3 Capturing Diagnostics ... 458
8.2.4 Reading Source Files from Memory ... 458
8.2.5 Writing Byte Codes to Memory ... 459
8.2.6 An Example: Dynamic Java Code Generation ... 461
8.3 Using Annotations ... 467
8.3.1 An Introduction into Annotations ... 468
8.3.2 An Example: Annotating Event Handlers ... 469
8.4 Annotation Syntax ... 475
8.4.1 Annotation Interfaces ... 475
8.4.2 Annotations ... 477
8.4.3 Annotating Declarations ... 479
8.4.4 Annotating Type Uses ... 480
8.4.5 Annotating `this` ... 481

8.5 Standard Annotations 482
8.5.1 Annotations for Compilation 483
8.5.2 Meta-Annotations 484
8.6 Source-Level Annotation Processing 488
8.6.1 Annotation Processors 488
8.6.2 The Language Model API 489
8.6.3 Using Annotations to Generate Source Code 490
8.7 Bytecode Engineering 493
8.7.1 Modifying Class Files 493
8.7.2 Modifying Bytecodes at Load Time 499

Chapter 9: The Java Platform Module System 503

9.1 The Module Concept 504
9.2 Naming Modules 505
9.3 The Modular "Hello, World!" Program 506
9.4 Requiring Modules 508
9.5 Exporting Packages 510
9.6 Modular JARs 514
9.7 Modules and Reflective Access 515
9.8 Automatic Modules 518
9.9 The Unnamed Module 521
9.10 Command-Line Flags for Migration 521
9.11 Transitive and Static Requirements 523
9.12 Qualified Exporting and Opening 525
9.13 Service Loading 526
9.14 Tools for Working with Modules 528

Chapter 10: Security 533

10.1 Class Loaders 534
10.1.1 The Class-Loading Process 534
10.1.2 The Class Loader Hierarchy 536
10.1.3 Using Class Loaders as Namespaces 538
10.1.4 Writing Your Own Class Loader 539
10.1.5 Bytecode Verification 545
10.2 User Authentication 549
10.2.1 The JAAS Framework 550
10.2.2 JAAS Login Modules 553

10.3 Digital Signatures 562
10.3.1 Message Digests 563
10.3.2 Message Signing 566
10.3.3 Verifying a Signature 569
10.3.4 The Authentication Problem 572
10.3.5 Certificate Signing 574
10.3.6 Certificate Requests 575
10.3.7 Code Signing 577
10.4 Encryption 578
10.4.1 Symmetric Ciphers 579
10.4.2 Key Generation 580
10.4.3 Cipher Streams 585
10.4.4 Public Key Ciphers 587

Chapter 11: Advanced Swing and Graphics 591

11.1 Tables 591
11.1.1 A Simple Table 592
11.1.2 Table Models 596
11.1.3 Working with Rows and Columns 600
11.1.3.1 Column Classes 600
11.1.3.2 Accessing Table Columns 601
11.1.3.3 Resizing Columns 602
11.1.3.4 Resizing Rows 603
11.1.3.5 Selecting Rows, Columns, and Cells 604
11.1.3.6 Sorting Rows 605
11.1.3.7 Filtering Rows 606
11.1.3.8 Hiding and Displaying Columns 608
11.1.4 Cell Rendering and Editing 617
11.1.4.1 Rendering Cells 617
11.1.4.2 Rendering the Header 619
11.1.4.3 Editing Cells 619
11.1.4.4 Custom Editors 621
11.2 Trees 630
11.2.1 Simple Trees 632
11.2.1.1 Editing Trees and Tree Paths 641
11.2.2 Node Enumeration 649

11.2.3 Rendering Nodes 651
11.2.4 Listening to Tree Events 654
11.2.5 Custom Tree Models 662
11.3 Advanced AWT 671
11.3.1 The Rendering Pipeline 672
11.3.2 Shapes 674
11.3.2.1 The Shape Class Hierarchy 675
11.3.2.2 Using the Shape Classes 677
11.3.3 Areas 691
11.3.4 Strokes 693
11.3.5 Paint 701
11.3.6 Coordinate Transformations 703
11.3.7 Clipping 710
11.3.8 Transparency and Composition 712
11.4 Raster Images 721
11.4.1 Readers and Writers for Images 722
11.4.1.1 Obtaining Readers and Writers for Image File Types 722
11.4.1.2 Reading and Writing Files with Multiple Images 724
11.4.2 Image Manipulation 732
11.4.2.1 Constructing Raster Images 733
11.4.2.2 Filtering Images 740
11.5 Printing 749
11.5.1 Graphics Printing 749
11.5.2 Multiple-Page Printing 759
11.5.3 Print Services 769
11.5.4 Stream Print Services 772
11.5.5 Printing Attributes 776

Chapter 12: Native Methods 785

12.1 Calling a C Function from a Java Program 786
12.2 Numeric Parameters and Return Values 793
12.3 String Parameters 795
12.4 Accessing Fields 801
12.4.1 Accessing Instance Fields 801

12.4.2 Accessing Static Fields 805
12.5 Encoding Signatures 806
12.6 Calling Java Methods 808
12.6.1 Instance Methods 809
12.6.2 Static Methods 810
12.6.3 Constructors 811
12.6.4 Alternative Method Invocations 811
12.7 Accessing Array Elements 816
12.8 Handling Errors 819
12.9 Using the Invocation API 825
12.10 A Complete Example: Accessing the Windows Registry 830
12.10.1 Overview of the Windows Registry 830
12.10.2 A Java Platform Interface for Accessing the Registry 832
12.10.3 Implementation of Registry Access Functions as Native Methods 833
12.11 Foreign Functions: A Glimpse into the Future 846

Index 849

Preface

To the Reader

The book you have in your hands is the second volume of the twelfth edition of *Core Java*, fully updated for Java 17. The first volume covers the essential features of the language; this volume deals with the advanced topics that a programmer needs to know for professional software development. Thus, as with the first volume and the previous editions of this book, we are still targeting programmers who want to put Java technology to work in real projects.

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

About This Book

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

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

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

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

Chapter 4 covers the networking API. Java makes it phenomenally easy to do complex network programming. We show you how to make network connections to servers, how to implement your own servers, and how to make HTTP connections. This chapter includes coverage of the new HTTP client.

Chapter 5 covers database programming. The focus is on JDBC, the Java database connectivity API that lets Java programs connect to relational databases. We show you how to write useful programs to handle realistic database chores, using a core subset of the JDBC API. (A complete treatment of the JDBC API would require a book almost as big as this one.)

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

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

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

In **Chapter 9**, you will learn about the Java Platform Module System that was introduced in Java 9 to facilitate an orderly evolution of the Java platform and core libraries. This module system provides encapsulation for packages and a mechanism for describing module requirements. You will learn the properties of modules so that you can decide whether to use them in your own applications. Even if you decide not to, you need to know the new rules so that you can interact with the Java platform and other modularized libraries.

Chapter 10 takes up the Java security model, user authentication, and the cryptographic functions in the Java security library. You will learn about important features such as message and code signing, authorization and authentication, and encryption. We conclude with examples that use the AES and RSA encryption algorithms.

Chapter 11 contains all the Swing material that didn't make it into Volume I, especially the important but complex tree and table components. We also cover the Java 2D API, which you can use to create realistic drawings and special effects. Of course, not many programmers need to program Swing user interfaces these days, so we pay particular attention to features that are useful for images that can be generated on a server.

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

As always, all chapters have been completely revised for the latest version of Java. Outdated material has been removed, and the new APIs up to Java 17 are covered in detail.

Conventions

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

NOTE: Notes are tagged with "note" icons that look like this.

TIP: Tips are tagged with “tip” icons that look like this.

CAUTION: When there is danger ahead, we warn you with a “caution” icon.

C++ NOTE: There are a number of C++ notes that explain the difference between the Java programming language and C++. You can skip them if you aren’t interested in C++.

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

`Application Programming Interface` 1.2

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

Listing 1.1 `ScriptTest.java`

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

Register your copy of *Core Java, Volume II: Advanced Features, Twelfth Edition,* on the InformIT site for convenient access to updates and/or corrections as they become available. To start the registration process, go to `informit.com/register` and log in or create an account. Enter the product ISBN (9780137871070) and click Submit. Look on the Registered Products tab for an Access Bonus Content link next to this product, and follow that link to access any available bonus materials. If you would like to be notified of exclusive offers on new editions and updates, please check the box to receive email from us.

Acknowledgments

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

A large number of individuals at Pearson 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 (Utah Valley University), Lance Andersen (Oracle), Gail Anderson (Anderson Software Group), Paul Anderson (Anderson Software Group), Alec Beaton (IBM), Cliff Berg, Andrew Binstock (Oracle), 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 (Clarity Training), Jim Gish (Oracle), Brian Goetz (Oracle), Angela Gordon, Dan Gordon (Electric Cloud), Rob Gordon, John Gray (University of Hartford), Cameron Gregory (olabs.com), Steve Haines, Marty Hall (coreservlets.com, Inc.), Vincent Hardy (Adobe Systems), Dan Harkey (San Jose State University), William Higgins (IBM), Marc Hoffmann (mtrail), Vladimir Ivanovic (PointBase), Jerry Jackson (CA Technologies), Heinz Kabutz (Java Specialists), Stepan V. Kalinin (I-Teco/Servionica LTD), Tim Kimmet (Walmart), Chris Laffra, Charlie Lai (Apple), Angelika Langer, Jeff Langr (Langr Software Solutions), 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), José Paumard (Oracle), Hao Pham, Paul Philion, Blake Ragsdell, Ylber Ramadani (Ryerson University), Stuart Reges (University of Arizona), Simon Ritter (Azul Systems), 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 (Sun Microsystems), Yoshiki Shibata, Richard Slywczak (NASA/Glenn Research Center), Bradley A. Smith, Steven Stelting (Oracle), Christopher Taylor, Luke Taylor (Valtech), George Thiruvathukal, Kim Topley (StreamingEdge), Janet Traub, Paul Tyma (consultant), Christian Ullenboom, Peter van der Linden, Burt Walsh, Joe Wang (Oracle), Dan Xu (Oracle), and John Zavgren (Oracle).

Cay Horstmann
Berlin, Germany
January 2022

CHAPTER 9

The Java Platform Module System

In this chapter

- 9.1 The Module Concept, page 504
- 9.2 Naming Modules, page 505
- 9.3 The Modular “Hello, World!” Program, page 506
- 9.4 Requiring Modules, page 508
- 9.5 Exporting Packages, page 510
- 9.6 Modular JARs, page 514
- 9.7 Modules and Reflective Access, page 515
- 9.8 Automatic Modules, page 518
- 9.9 The Unnamed Module, page 521
- 9.10 Command-Line Flags for Migration, page 521
- 9.11 Transitive and Static Requirements, page 523
- 9.12 Qualified Exporting and Opening, page 525
- 9.13 Service Loading, page 526
- 9.14 Tools for Working with Modules, page 528

An important characteristic of object-oriented programming is encapsulation. A class declaration consists of a public interface and a private implementation. A class can evolve by changing the implementation without affecting its users. A module system provides the same benefits for programming in the large. A module can make classes and packages selectively available so that its evolution can be controlled.

Several existing Java module systems rely on class loaders to isolate classes. However, Java 9 introduced a new system, called the Java Platform Module System, that is supported by the Java compiler and virtual machine. It was designed to modularize the large code base of the Java platform. You can, if you choose, use this system to modularize your own applications.

Whether or not you use Java platform modules in your own applications, you may be impacted by the modularized Java platform. This chapter shows you how to declare and use Java platform modules. You will also learn how to migrate your applications to work with the modularized Java platform and third-party modules.

9.1 The Module Concept

In object-oriented programming, the fundamental building block is the class. Classes provide encapsulation. Private features can only be accessed by code that has explicit permission—namely, the methods of the class. This makes it possible to reason about access. If a private variable has changed, you can produce a set of all possible culprits. If you need to modify the private representation, you know which methods are affected.

In Java, packages provide the next larger organizational grouping. A package is a collection of classes. Packages also provide a level of encapsulation. Any feature with package access (neither public nor private) is accessible only from methods in the same package.

However, in large systems, this level of access control is not enough. Any public feature (that is, a feature that is accessible outside a package) is accessible everywhere. Suppose you want to modify or drop a rarely used feature. Once it is public, there is no way to reason about the impact of that change.

This is the situation that the Java platform designers faced. Over twenty years, the JDK grew by leaps and bounds, but clearly some features are now essentially obsolete. Everyone's favorite example is CORBA. When was the last time you used it? Yet, the `org.omg.corba` package was shipped with every JDK until Java 10. As of Java 11, those few who still need it must add the required JAR files to their projects.

What about `java.awt`? It shouldn't be required in a server-side application, right? Except that the class `java.awt.DataFlavor` is used in the implementation of SOAP, an XML-based web services protocol.

The Java platform designers, faced with a giant hairball of code, decided that they needed a structuring mechanism that provides more control. They looked at existing module systems (such as OSGi) and found them unsuitable for their problem. Instead, they designed a new system, called the *Java Platform Module System*, that is now a part of the Java language and virtual machine. That system has been used successfully to modularize the Java API, and you can, if you so choose, use it with your own applications.

A Java platform module consists of

- A collection of packages
- Optionally, resource files and other files such as native libraries
- A list of the accessible packages in the module
- A list of all modules on which this module depends

The Java platform enforces encapsulation and dependencies, both at compile time and in the virtual machine.

Why should you consider using the Java Platform Module System for your own programs instead of following the traditional approach of using JAR files on the class path? There are two advantages.

1. Strong encapsulation: You can control which of your packages are accessible, and you don't have to worry about maintaining code that you didn't intend for public consumption.
2. Reliable configuration: You avoid common class path problems such as duplicate or missing classes.

There are some issues that the Java Platform Module System does not address, such as versioning of modules. There is no support for specifying which version of a module is required, or for using multiple versions of a module in the same program. These can be desirable features, but you must use mechanisms other than the Java Platform Module System if you need them.

9.2 Naming Modules

A module is a collection of packages. The package names in the module need not be related. For example, the module `java.sql` contains packages `java.sql`, `javax.sql`, and `javax.transaction.xa`. Also, as you can see from this example, it is perfectly acceptable for the module name to be the same as a package name.

Just like a path name, a module name is made up of letters, digits, underscores, and periods. Also, just as with path names, there is no hierarchical relationship between modules. If you had a module `com.horstmann` and another module `com.horstmann.corejava`, they would be unrelated, as far as the module system is concerned.

When creating a module for use by others, it is important to ensure that its name is globally unique. It is expected that most module names will follow the "reverse domain name" convention, just like package names.

The easiest approach is to name a module after the top-level package that the module provides. For example, the SLF4J logging façade has a module `org.slf4j` with packages `org.slf4j`, `org.slf4j.spi`, `org.slf4j.event`, and `org.slf4j.helpers`.

This convention prevents package name conflicts in modules. Any given package can only be placed in one module. If your module names are unique and your package names start with the module name, then your package names will also be unique.

You can use shorter module names for modules that are not meant to be used by other programmers, such as a module containing an application program. Just to show that it can be done, I will do the same in this chapter. Modules with what could plausibly be library code will have names such as `com.horstmann.util`, and modules containing programs (with a class that has a `main` method) will have catchy names such as `v2ch09.hellomod`.

NOTE: Module names are only used in module declarations. In the source files for your Java classes, you never refer to module names; instead, use package names the way they have always been used.

9.3 The Modular "Hello, World!" Program

Let us put the traditional "Hello, World!" program into a module. First, we need to put the class into a package—the "unnamed package" cannot be contained in a module. Here it is:

```
package com.horstmann.hello;

public class HelloWorld
{
   public static void main(String[] args)
   {
      System.out.println("Hello, Modular World!");
   }
}
```

So far, nothing has changed. To make a module `v2ch09.hellomod` containing this package, you need to add a module declaration. You place it in a file named `module-info.java`, located in the base directory (that is, the same directory that contains the `com` directory). By convention, the name of the base directory is the same as the module name.

```
v2ch09.hellomod/
└ module-info.java
  com/
  └ horstmann/
    └ hello/
      └ HelloWorld.java
```

The `module-info.java` file contains the module declaration:

```
module v2ch09.hellomod
{
}
```

This module declaration is empty because the module has nothing to offer to anyone, nor does it need anything.

Now, compile as usual:

```
javac v2ch09.hellomod/module-info.java v2ch09.hellomod/com/horstmann/hello/HelloWorld.java
```

The `module-info.java` file doesn't look like a Java source file, and of course there can't be a class with the name `module-info`, since class names cannot contain hyphens. The `module` keyword, as well as keywords `requires`, `exports`, and so on, that you will see in the following sections, are "restricted keywords" that have a special meaning only in module declarations. The file is compiled into a class file `module-info.class` that contains the module definition in binary form.

To run this program as a modular application, you specify the *module path*, which is similar to the class path but contains modules. You also specify the main class in the format *modulename*/*classname*:

```
java --module-path v2ch09.hellomod --module v2ch09.hellomod/com.horstmann.hello.HelloWorld
```

Instead of `--module-path` and `--module`, you can use the single-letter options `-p` and `-m`:

```
java -p v2ch09.hellomod -m v2ch09.hellomod/com.horstmann.hello.HelloWorld
```

Either way, the `"Hello, Modular World!"` greeting will appear, demonstrating that you have successfully modularized your first application.

NOTE: When you compile this module, you get a warning:

```
warning: [module] module name component v2ch09 should avoid terminal digits
```

This warning is intended to discourage programmers from adding version numbers to module names. You can ignore the warning, or suppress it with an annotation:

```
@SuppressWarnings("module")
module v2ch09.hellomod
{
}
```

In this one respect, the `module` declaration is just like a class declaration: You can annotate it. (The annotation type must have target `ElementType.MODULE`.)

9.4 Requiring Modules

Let us make a new module `v2ch09.requiremod` in which a program uses a `JOptionPane` to show the "Hello, Modular World!" message:

```
package com.horstmann.hello;

import javax.swing.JOptionPane;

public class HelloWorld
{
   public static void main(String[] args)
   {
      JOptionPane.showMessageDialog(null, "Hello, Modular World!");
   }
}
```

Now compilation fails with this message:

```
error: package javax.swing is not visible
  (package javax.swing is declared in module java.desktop,
  but module v2ch09.requiremod does not read it)
```

The JDK has been modularized, and the `javax.swing` package is now contained in the `java.desktop` module. Our module needs to declare that it relies on that module:

```
module v2ch09.requiremod
{
   requires java.desktop;
}
```

It is a design goal of the module system that modules are explicit about their requirements, so the virtual machine can ensure that all requirements are fulfilled before starting a program.

In the preceding section, the need for explicit requirements did not arise because we only used the `java.lang` and `java.io` packages. These packages are included in the `java.base` module which is required by default.

Note that our `v2ch09.requiremod` module lists only its own module requirements. It requires the `java.desktop` module so that it can use the `javax.swing` package. The `java.desktop` module itself declares that it requires three other modules, namely `java.datatransfer`, `java.prefs`, and `java.xml`.

Figure 9.1 shows the *module graph* whose nodes are modules. The edges of the graph—the arrows joining nodes—are either declared requirements or the implied requirement on `java.base` when none is declared.

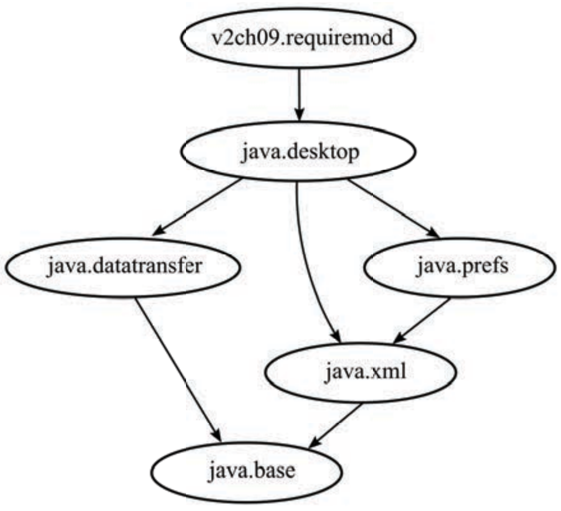


Figure 9.1 The module graph of the Swing "Hello, Modular World" application

You cannot have cycles in the module graph—that is, a module cannot directly or indirectly require itself.

A module does not automatically pass on access rights to other modules. In our example, the `java.desktop` module declares that it requires `java.prefs`, and the `java.prefs` module declares that it requires `java.xml`. That does not give `java.desktop` the right to use packages from the `java.xml` module. It needs to explicitly declare that requirement. In mathematical terms, the `requires` relationship is not "transitive." Generally, this behavior is desirable because it makes requirements explicit, but as you will see in Section 9.11, "Transitive and Static Requirements," on p. 523, you can relax it in some cases.

NOTE: The error message at the beginning of this section stated that our `v2ch09.requiremod` module did not "read" the `java.desktop` module. In the parlance of the Java Platform Module System, module *M reads* module *N* in the following cases:

1. *M* requires *N*.
2. *M* requires a module that transitively requires *N* (see Section 9.11, "Transitive and Static Requirements," on p. 523).
3. *N* is *M* or `java.base`.

9.5 Exporting Packages

In the preceding section, you saw that a module must require another module if it wants to use its packages. However, that does not automatically make all packages in the required module available. A module states which of its packages are accessible, using the `exports` keyword. For example, here is a part of the module declaration for the `java.xml` module:

```
module java.xml
{
   exports javax.xml;
   exports javax.xml.catalog;
   exports javax.xml.datatype;
   exports javax.xml.namespace;
   exports javax.xml.parsers;
   . . .
}
```

This module makes many packages available, but hides others (such as `jdk.xml.internal`) by not exporting them.

When a package is exported, its `public` and `protected` classes and interfaces, and their `public` and `protected` members, are accessible outside the module. (As always, `protected` types and members are accessible only in subclasses.)

However, a package that is not exported is not accessible outside its own module. This is quite different from Java before modules. In the past, you were able to use public classes from any package, even if it was not part of the public API. For example, it was commonly recommended to use classes such as `sun.misc.BASE64Encoder` or `com.sun.rowset.CachedRowSetImpl` when the public API did not provide the appropriate functionality.

Nowadays, you can no longer access unexported packages from the Java platform API since all of them are contained inside modules. As a result, some programs will no longer run with Java 9. Of course, nobody ever committed to keeping non-public APIs available, so this should not come as a shock.

Let us put exports to use in a simple situation. We will prepare a module `com.horstmann.greet` that exports a package, also called `com.horstmann.greet`, following the convention that a module that provides code for others should be named after the top-level package inside it. There is also a package `com.horstmann.greet.internal` that we don't export.

A public `Greeter` interface is in the first package.

```
package com.horstmann.greet;

public interface Greeter
{
   static Greeter newInstance()
   {
      return new com.horstmann.greet.internal.GreeterImpl();
   }

   String greet(String subject);
}
```

The second package has a class that implements the interface. The class is public since it is accessed in the first package.

```
package com.horstmann.greet.internal;

import com.horstmann.greet.Greeter;

public class GreeterImpl implements Greeter
{
   public String greet(String subject)
   {
      return "Hello, " + subject + "!";
   }
}
```

The `com.horstmann.greet` module contains both packages but only exports the first:

```
module com.horstmann.greet
{
   exports com.horstmann.greet;
}
```

The second package is inaccessible outside the module.

We put our application into a second module, which will require the first module:

```
module v2ch09.exportedpkg
{
   requires com.horstmann.greet;
}
```

NOTE: The `exports` statement is followed by a package name, whereas `requires` is followed by a module name.

Our application now uses a `Greeter` to obtain a greeting:

```
package com.horstmann.hello;

import com.horstmann.greet.Greeter;

public class HelloWorld
{
   public static void main(String[] args)
   {
      Greeter greeter = Greeter.newInstance();
      System.out.println(greeter.greet("Modular World"));
   }
}
```

Here is the source file structure for these two modules:

```
com.horstmann.greet
├ module-info.java
└ com
  └ horstmann
    └ greet
      ├ Greeter.java
      └ internal
        └ GreeterImpl.java
v2ch09.exportedpkg
├ module-info.java
└ com
  └ horstmann
    └ hello
      └ HelloWorld.java
```

To build this application, first compile the `com.horstmann.greet` module:

```
javac com.horstmann.greet/module-info.java \
   com.horstmann.greet/com/horstmann/greet/Greeter.java \
   com.horstmann.greet/com/horstmann/greet/internal/GreeterImpl.java
```

Then compile the application module with the first module on the module path:

```
javac -p com.horstmann.greet v2ch09.exportedpkg/module-info.java \
   v2ch09.exportedpkg/com/horstmann/hello/HelloWorld.java
```

Finally, run the program with both modules on the module path:

```
java -p v2ch09.exportedpkg:com.horstmann.greet \
   -m v2ch09.exportedpkg/com.horstmann.hello.HelloWorld
```

TIP: To build this application with Eclipse, make a separate project for each module. In the `v2ch09.exportedpkg` project, edit the project properties. In the Projects tab, add the `com.horstmann.greet` module to the module path—see Figure 9.2.

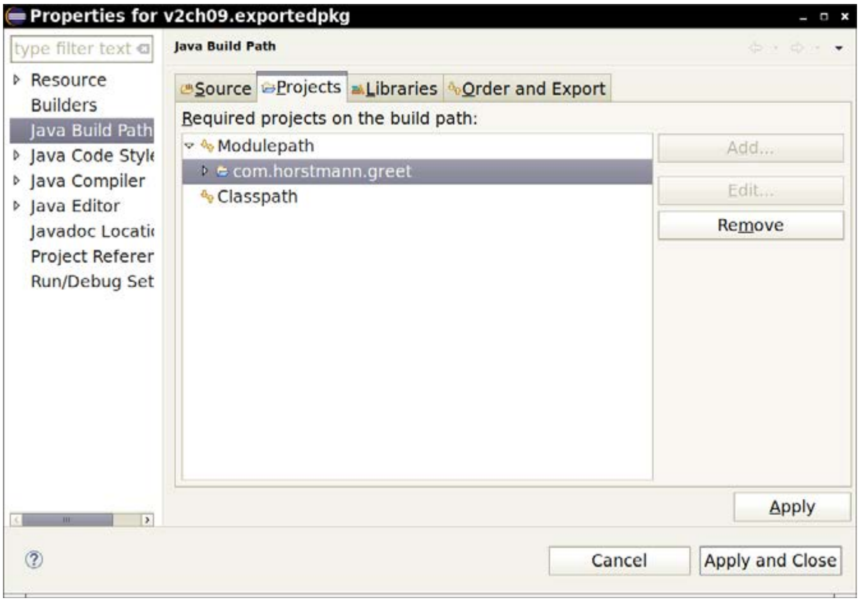


Figure 9.2 Adding a dependent module to an Eclipse project

You have now seen the `requires` and `exports` statements that form the backbone of the Java Platform Module System. As you can see, the module system is conceptually simple. Modules specify what modules they need, and which packages they offer to other modules. Section 9.12, "Qualified Exporting and Opening," on p. 525 shows a minor variation of the `exports` statement.

CAUTION: A module does not provide a scope. You cannot have two packages with the same name in different modules. This is true even for hidden packages (that is, packages that are not exported.)

9.6 Modular JARs

So far, we have simply compiled modules into the directory tree of the source code. Clearly, that is not satisfactory for deployment. Instead, a module can be deployed by placing all its classes in a JAR file, with a `module-info.class` in the root. Such a JAR file is called a *modular* JAR.

To create a modular JAR file, use the `jar` tool in the usual way. If you have multiple packages, it is best to compile with the `-d` option which places class files into a separate directory. The directory is created if it doesn't already exists. Then use the `-C` option of the `jar` command to change to that directory when collecting files.

```
javac -d modules/com.horstmann.greet $(find com.horstmann.greet -name *.java)
jar -c -v -f com.horstmann.greet.jar -C modules/com.horstmann.greet .
```

If you use a build tool such as Maven, Ant, or Gradle, just keep building your JAR file as you always do. As long as `module-info.class` is included, you get a modular JAR.

Then, include the modular JAR in the module path, and the module will be loaded.

CAUTION: In the past, classes of a package were sometimes distributed over multiple JAR files. (Such a package is called a "split package".) This was probably never a good idea, and it is not possible with modules.

As with regular JAR files, you can specify a main class in a modular JAR:

```
javac -p com.horstmann.greet.jar \
   -d modules/v2ch09.exportedpkg $(find v2ch09.exportedpkg -name *.java)
jar -c -v -f v2ch09.exportedpkg.jar -e com.horstmann.hello.HelloWorld \
   -C modules/v2ch09.exportedpkg .
```

When you launch the program, you specify the module containing the main class:

```
java -p com.horstmann.greet.jar:v2ch09.exportedpkg.jar -m v2ch09.exportedpkg
```

When creating a JAR file, you can optionally specify a version number. Use the `--module-version` parameter, and also add `@` and the version number to the JAR file name:

```
jar -c -v -f com.horstmann.greet@1.0.jar --module-version 1.0 -C com.horstmann.greet .
```

As already discussed, the version number is not used by the Java Platform Module System for resolving modules, but it can be queried by other tools and frameworks.

NOTE: You can find out the version number through the reflection API. In our example:

```
Optional<String> version = Greeter.class.getModule().getDescriptor().rawVersion();
```

yields an `Optional` containing the version string `"1.0"`.

NOTE: The module equivalent to a class loader is a *layer*. The Java Platform Module System loads the JDK modules and application modules into the *boot layer*. A program can load other modules, using the layer API (which is not covered in this book). Such a program may choose to take module versions into account. It is expected that developers of programs such as Java EE application servers will make use of the layer API to provide support for modules.

TIP: If you want to load a module into JShell, include the JAR on the module path and use the `--add-modules` option:

```
jshell --module-path com.horstmann.greet@1.0.jar --add-modules com.horstmann.greet
```

9.7 Modules and Reflective Access

In the preceding sections, you saw that the module system enforces encapsulation. A module can only access explicitly exported packages from another module. In the past, it was always possible to overcome pesky access restrictions by using reflection. As you have seen in Chapter 5 of Volume I, reflection can access private members of any class.

However, in the modular world, that is no longer true. If a class is inside a module, reflective access to non-public members will fail. Specifically, recall how we accessed private fields:

```
Field f = obj.getClass().getDeclaredField("salary");
f.setAccessible(true);
```

```
double value = f.getDouble(obj);
f.setDouble(obj, value * 1.1);
```

The call `f.setAccessible(true)` succeeds unless a security manager disallows private field access. However, it is not common to run Java applications with security managers, and there are many libraries that use reflective access. Typical examples are object-relational mappers, such as JPA, that automatically persist objects in databases and libraries that convert between objects and XML or JSON, such as JAXB and JSON-B.

If you use such a library, and you also want to use modules, you have to be careful. To demonstrate this issue, let us place the `ObjectAnalyzer` class from Chapter 5 of Volume I into a module `com.horstmann.util`. That class has a `toString` method that prints the fields of an object, using reflection.

A separate `v2ch09.openpkg` module contains a simple `Country` class:

```
package com.horstmann.places;

public class Country
{
   private String name;
   private double area;

   public Country(String name, double area)
   {
      this.name = name;
      this.area = area;
   }
   // . . .
}
```

A short program demonstrates how to analyze a `Country` object:

```
package com.horstmann.places;

import com.horstmann.util.*;

public class Demo
{
   public static void main(String[] args) throws ReflectiveOperationException
   {
      var belgium = new Country("Belgium", 30510);
      var analyzer = new ObjectAnalyzer();
      System.out.println(analyzer.toString(belgium));
   }
}
```

Now compile both modules and the `Demo` program:

```
javac com.horstmann.util/module-info.java \
   com.horstmann.util/com/horstmann/util/ObjectAnalyzer.java
javac -p com.horstmann.util v2ch09.openpkg/module-info.java \
   v2ch09.openpkg/com/horstmann/places/*.java
java -p v2ch09.openpkg:com.horstmann.util -m v2ch09.openpkg/com.horstmann.places.Demo
```

The program will fail with an exception:

```
Exception in thread "main" java.lang.reflect.InaccessibleObjectException:
   Unable to make field private java.lang.String com.horstmann.places.Country.name
   accessible: module v2ch09.openpkg does not "opens com.horstmann.places" to module
   com.horstmann.util
```

Of course, in pristine theory, it is wrong to violate encapsulation and poke around in the private members of an object. But mechanisms such as object-relational mapping or XML/JSON binding are so common that the module system must accommodate them.

Using the `opens` keyword, a module can *open* a package, which enables reflective access to all instances of classes in the given package. Here is what our module has to do:

```
module v2ch09.openpkg
{
   requires com.horstmann.util;
   opens com.horstmann.places;
}
```

With this change, the `ObjectAnalyzer` will work correctly.

A module can be declared as `open`, such as

```
open module v2ch09.openpkg
{
   requires com.horstmann.util;
}
```

An open module grants runtime access to all of its packages, as if all packages had been declared with `exports` and `opens`. However, only explicitly exported packages are accessible at compile time. Open modules combine the compile-time safety of the module system with the classic permissive runtime behavior.

Recall from Chapter 5 of Volume I that JAR files can contain, in addition to class files and a manifest, *file resources* which can be loaded with the method `Class.getResourceAsStream`, and now also with `Module.getResourceAsStream`. If a resource is stored in a directory that matches a package in a module, then the package must be opened to the caller. Resources in other directories, as well as the class files and manifest, can be read by anyone.

NOTE: For a more realistic example, we can convert the `Country` object to XML or JSON, using the JSON-B specification. To use the Yasson implementation of JSON-B, download the JAR files `jakarta.json-api-2.0.1.jar`, `jakarta.json.bind-api-2.0.0.jar`, `jakarta.json-2.0.1-module.jar`, and `yasson-2.0.3.jar` from the Maven Central Repository. Place the JAR files on the module path and run the `com.horstmann.places.Demo` program in the `v2ch09.openpkg2` module. When the `com.horstmann.places` package is opened, conversion to JSON succeeds.

NOTE: It is possible that future libraries will use *variable handles* instead of reflection for reading and writing fields. A `VarHandle` is similar to a `Field`. You can use it to read or write a specific field of any instance of a specific class. However, to obtain a `VarHandle`, the library code needs a `Lookup` object:

```
public Object getFieldValue(Object obj, String fieldName, Lookup lookup)
      throws NoSuchFieldException, IllegalAccessException
{
   Class<?> cl = obj.getClass();
   Field field = cl.getDeclaredField(fieldName);
   VarHandle handle = MethodHandles.privateLookupIn(cl, lookup)
      .unreflectVarHandle(field);
   return handle.get(obj);
}
```

This works provided the `Lookup` object is generated in the module that has the permission to access the field. Some method in the module simply calls `MethodHandles.lookup()`, which yields an object encapsulating the access rights of the caller. In this way, one module can give permission for accessing private members to another module. The practical issue is how those permissions can be given with a minimum of hassle.

9.8 Automatic Modules

You now know to put the Java Platform Module System to use. If you start with a brand-new project in which you write all the code yourself, you can design modules, declare module dependencies, and package your application into modular JAR files.

However, that is an extremely uncommon scenario. Almost all projects rely on third-party libraries. Of course, you can wait until the providers of all libraries have turned them into modules, and then modularize your own code.

But what if you don't want to wait? The Java Platform Module System provides two mechanisms for crossing the chasm that separates today's premodular world and fully modular applications: automatic modules and the unnamed module.

For migration purposes, you can turn any JAR file into a module simply by placing it onto a directory in the module path instead of the class path. A JAR without a `module-info.class` on the module path is called an *automatic module*. An automatic module has the following properties:

1. The module implicitly has a `requires` clause for all other modules.
2. All of its packages are exported and opened.
3. If there is an entry with key `Automatic-Module-Name` in the JAR file manifest `META-INF/MANIFEST.MF`, its value becomes the module name.
4. Otherwise the module name is obtained from the JAR file name, dropping any trailing version number and replacing sequences of non-alphanumeric characters with a dot.

The first two rules imply that the packages in the automatic module act as if they were on the class path. The reason for using the module path is for the benefit of other modules, allowing them to express dependencies on this module.

Suppose, for example, that you are implementing a module that processes CSV files and uses the Apache Commons CSV library. You would like to express in your `module-info.java` file that your module depends on Apache Commons CSV.

If you add `commons-csv-1.9.0.jar` onto the module path, then your modules can reference the module. Its name is `commons.csv` since the trailing version number `-1.9.0` is removed and the non-alphanumeric character `-` is replaced by a dot.

This name might be an acceptable module name because Commons CSV is well known and it is unlikely that someone else will try to use the same name for a different module. But it would be better if the maintainers of this JAR file could quickly agree to reserve a reverse DNS name, preferably the top-level package name `org.apache.commons.csv`, as the module name. They just need to add a line

```
Automatic-Module-Name: org.apache.commons.csv
```

to the `META-INF/MANIFEST.MF` file inside the JAR. Eventually, hopefully, they will turn the JAR file into a true module by adding `module-info.java` with the reserved module name—and every other module that refers to the CSV module with that name will just continue to work.

NOTE: The migration plan to modules is a great social experiment, and nobody knows whether it will end well. Before you put third-party JARs on the module path, check whether they are modular, and if not, whether their manifest has a module name. If not, you can still turn the JAR into an automatic module, but be prepared to update the module name later.

As this book is being written, version 1.9.0 of the Commons CSV JAR file does not have a module descriptor or an automatic module name. Nevertheless, it will work fine on the module path. You can download the library from `https://commons.apache.org/proper/commons-csv`. Place the `commons-csv-1.9.0.jar` file into the directory of the `v2ch9.automod` module. That module contains a simple program that reads a CSV file with country data:

```
package com.horstmann.places;

import java.io.*;
import org.apache.commons.csv.*;

public class CSVDemo
{
   public static void main(String[] args) throws IOException
   {
      var in = new FileReader("countries.csv");
      Iterable<CSVRecord> records = CSVFormat.EXCEL.withDelimiter(';')
            .withHeader().parse(in);
      for (CSVRecord record : records)
      {
         String name = record.get("Name");
         double area = Double.parseDouble(record.get("Area"));
         System.out.println(name + " has area " + area);
      }
   }
}
```

Since we will use `commons-csv-1.9.0.jar` as an automatic module, we need to require it:

```
@SuppressWarnings("module")
module v2ch09.automod
{
   requires commons.csv;
}
```

Here are the commands for compiling and running the program:

```
javac -p v2ch09.automod:commons-csv-1.9.0.jar \
   v2ch09.automod/com/horstmann/places/CSVDemo.java \
   v2ch09.automod/module-info.java
```

```
java -p v2ch09.automod:commons-csv-1.9.0.jar \
   -m v2ch09.automod/com.horstmann.places.CSVDemo
```

9.9 The Unnamed Module

Any class that is not on the module path is part of an *unnamed module*. Technically, there may be more than one unnamed module, but all of them together act as if they are a single module which is called *the* unnamed module. As with automatic modules, the unnamed module can access all other modules, and all of its packages are exported and opened.

However, *no explicit module* can access the unnamed module. (An explicit module is a module that is neither automatic nor unnamed—that is, a module with a `module-info.class` on the module path.) In other words, explicit modules are always free from the "class path hell."

Consider, for example, the program of the preceding section. Suppose you put `commons-csv-1.9.0.jar` onto the class path instead of the module path:

```
java --module-path v2ch09.automod \
   --class-path commons-csv-1.9.0.jar \
   -m v2ch09.automod/com.horstmann.places.CSVDemo
```

Now the program won't start:

```
Error occurred during initialization of boot layer
java.lang.module.FindException: Module commons.csv not found, required by v2ch09.automod
```

Therefore, migration to the Java Platform Module System is necessarily a bottom-up process:

1. The Java platform itself is modularized.
2. Next, libraries are modularized, either by using automatic modules or by turning them into explicit modules.
3. Once all libraries used by your application are modularized, you can turn the code of your application into a module.

NOTE: Automatic modules *can* read the unnamed module, so their dependencies can go onto the class path.

9.10 Command-Line Flags for Migration

Even if your programs do not use modules, you cannot escape the modular world when using Java 9 and beyond. Your application code may reside on the class path in an unnamed module, so that all packages are exported and opened. Still, the code interacts with the Java platform, which is modularized.

As of Java 11, compile-time encapsulation is strictly enforced. However, before Java 16, runtime access was permitted. The default behavior was to display a warning on the console for the first instance of each offense. As of Java 16, reflective access at runtime is also enforced. In order to give you time to prepare for that change, the `java` launcher in Java 9 through 16 had an `--illegal-access` flag with four possible settings:

1. `--illegal-access=permit` is the Java 9 default behavior, printing a message for the first instance of illegal access.
2. `--illegal-access=warn` prints a message for each illegal access.
3. `--illegal-access=debug` prints a message and stack trace for each illegal access.
4. `--illegal-access=deny` is the Java 16 default behavior, denying all illegal access.

The `--illegal-access` flag is no longer usable in Java 17.

The `--add-exports` and `--add-opens` flags allow you to tweak legacy applications. Consider an application that uses an internal API which is no longer accessible, such as `com.sun.rowset.CachedRowSetImpl`. The best remedy is to change the implementation. (As of Java 7, you can get a cached row set from a `RowSetProvider`.) But suppose you don't have access to the source code.

In that case, start the application with the `--add-exports` flag. Specify the module and the package that you want to export, and the module to which you want to export the package, which in our case is the unnamed module.

```
java --add-exports java.sql.rowset/com.sun.rowset=ALL_UNNAMED \
   -jar MyApp.jar
```

Now, suppose your application uses reflection to access private fields or methods. Reflection inside the unnamed module is OK, but it is no longer possible to reflectively access non-public members of the Java platform classes. For example, some libraries that dynamically generate Java classes call the protected `ClassLoader.defineClass` method through reflection. If an application uses such a library, add the flag

```
--add-opens java.base/java.lang=ALL-UNNAMED
```

When adding all those command-line options to get a legacy app to work, you may well end up with the command line from hell. To better manage multiple options, you can put them in one or more files specified with an `@` prefix. For example,

```
java @options1 @options2 -jar MyProg.java
```

where the files `options1` and `options2` contain options for the `java` command.

There are a few syntax rules for the options files:

- Separate options with spaces, tabs, or newlines.
- Use double quotes around arguments that include spaces, such as `"Program Files"`.
- A line ending in a \ is merged with the next line.
- Backslashes must be escaped, such as `C:\\Users\\Fred`.
- Comment lines start with `#`.

9.11 Transitive and Static Requirements

In Section 9.4, "Requiring Modules," on p. 508, you have seen the basic form of the `requires` statement. In this section, you will see two variants that are occasionally useful.

In some situation, it can be tedious for a user of a given module to declare all required modules. Consider, for example, the `java.desktop` module. It requires three modules: `java.prefs`, `java.datatransfer` and `java.xml`. The `java.prefs` module is only used internally. However, classes from `java.datatransfer` and `java.xml` appear in the public API, in methods such as

```
java.awt.datatransfer.Clipboard java.awt.Toolkit.getSystemClipboard()
java.beans.XMLDecoder(org.xml.sax.InputSource is)
```

That is not something that a user of the `java.desktop` module should have to think about. For that reason, the `java.desktop` module declares the requirement with the `transitive` modifier:

```
module java.desktop
{
   requires java.prefs;
   requires transitive java.datatransfer;
   requires transitive java.xml;
   . . .
}
```

Any module that declares a requirement on `java.desktop` now automatically requires these two modules.

NOTE: Some programmers recommend that you should always use `requires transitive` when a package from another module is used in the public API. But that is not a requirement of the Java language. Consider, for example, the `java.sql` module:

```
module java.sql
{
   requires transitive java.logging;
   . . .
}
```

There is a single use of a package from the `java.logging` module in the entire `java.sql` API, namely the `java.sql.Driver.parentLogger` method that returns a `java.util.logging.Logger`. It would have been perfectly acceptable to not declare this module requirement as transitive. Then, those modules—and only those—who actually use that method would need to declare that they require `java.logging`.

One compelling use of the `requires transitive` statement is an *aggregator* module—a module with no packages and only transitive requirements. One such module is the `java.se` module, declared like this:

```
module java.se
{
   requires transitive java.compiler;
   requires transitive java.datatransfer;
   requires transitive java.desktop;
   . . .
   requires transitive java.sql;
   requires transitive java.sql.rowset;
   requires transitive java.xml;
   requires transitive java.xml.crypto;
}
```

A programmer who isn't interested in fine-grained module dependencies can simply require `java.se` and get all modules of the Java SE platform.

Finally, there is an uncommon `requires static` variant that declares that a module must be present at compile time but is optional at runtime. There are two use cases:

1. To access an annotation that is processed at compile time and declared in a different module.
2. To use a class in a different module if it is available, and otherwise do something else, such as:

```
try
{
   new oracle.jdbc.driver.OracleDriver();
   . . .
}
catch (NoClassDefFoundError er)
{
   Do something else
}
```

9.12 Qualified Exporting and Opening

In this section, you will see a variant of the `exports` and `opens` statement that narrows their scope to a specified set of modules. For example, the `java.base` module contains a statement

```
exports sun.net to
  java.net.http,
  jdk.naming.dns;
```

Such a statement is called a *qualified export*. The listed modules can access the exported package, but other modules cannot.

Excessive use of qualified exports can indicate a poor modular structure. Nevertheless, they can arise when modularizing an existing code base. Here, the `sun.net` package is placed inside the `java.base` module because that is where it is mostly needed. However, a couple of other modules also use that package. The Java platform designers didn't want to make `java.base` even bigger, and they didn't want to make the internal `sun.net` package generally available. In a greenfield project, one can instead design a more modular API.

Similarly, you can restrict the `opens` statement to specific modules. For example, in Section 9.7, "Modules and Reflective Access," on p. 515 we could have used a qualified `opens` statement, like this:

```
module v2ch09.openpkg
{
   requires com.horstmann.util;
   opens com.horstmann.places to com.horstmann.util;
}
```

Now the `com.horstmann.places` package is only opened to the `com.horstmann.util` module.

9.13 Service Loading

The ServiceLoader class (see Chapter 6 of Volume I) provides a lightweight mechanism for matching up service interfaces with implementations. The Java Platform Module System makes this mechanism easier to use.

Here is a quick reminder of service loading. A service has an interface and one or more possible implementations. Here is a simple example of an interface:

```
public interface GreeterService
{
   String greet(String subject);
   Locale getLocale();
}
```

One or more modules provide implementations, such as

```
public class FrenchGreeter implements GreeterService
{
   public String greet(String subject) { return "Bonjour " + subject; }
   public Locale getLocale() { return Locale.FRENCH; }
}
```

The service consumer must pick an implementation among all offered implementations, based on whatever criteria it deems appropriate.

```
ServiceLoader<GreeterService> greeterLoader = ServiceLoader.load(GreeterService.class);
GreeterService chosenGreeter;
for (GreeterService greeter : greeterLoader)
{
   if (. . .)
   {
      chosenGreeter = greeter;
   }
}
```

In the past, implementations were offered by placing text files into the META-INF/services directory of the JAR file containing the implementation classes. The module system provides a better approach. Instead of text files, you can add statements to the module descriptors.

A module providing an implementation of a service adds a provides statement that lists the service interface (which may be defined in any module) and the implementing class (which must be a part of this module). Here is an example from the jdk.security.auth module:

```
module jdk.security.auth
{
   . . .
   provides javax.security.auth.spi.LoginModule with
      com.sun.security.auth.module.Krb5LoginModule,
      com.sun.security.auth.module.UnixLoginModule,
      com.sun.security.auth.module.JndiLoginModule,
      com.sun.security.auth.module.KeyStoreLoginModule,
      com.sun.security.auth.module.LdapLoginModule,
      com.sun.security.auth.module.NTLoginModule;
}
```

This is the equivalent of the `META-INF/services` file.

A consuming module contains a `uses` statement.

```
module java.base
{
   . . .
   uses javax.security.auth.spi.LoginModule;
}
```

When code in a consuming module calls `ServiceLoader.load(`*ServiceInterface*`.class)`, the matching provider classes will be loaded, even though they may not be in accessible packages.

In our code example, we provide implementations for a German and French greeter in the package `com.horstmann.greetsvc.internal`. The service module exports the `com.horstmann.greetsvc` package, but not the package with the implementations. The `provides` statement declares the service and its implementing classes in the unexported package:

```
module com.horstmann.greetsvc
{
   exports com.horstmann.greetsvc;

   provides com.horstmann.greetsvc.GreeterService with
      com.horstmann.greetsvc.internal.FrenchGreeter,
      com.horstmann.greetsvc.internal.GermanGreeterFactory;
}
```

The `v2ch09.useservice` module consumes the service. Using the `ServiceLoader` facility, we iterate over the provided services and pick the one matching the desired language:

```
package com.horstmann.hello;

import java.util.*;
import com.horstmann.greetsvc.*;

public class HelloWorld
{
   public static void main(String[] args)
   {
      ServiceLoader<GreeterService> greeterLoader
            = ServiceLoader.load(GreeterService.class);
      String desiredLanguage = args.length > 0 ? args[0] : "de";
      GreeterService chosenGreeter = null;
      for (GreeterService greeter : greeterLoader)
      {
         if (greeter.getLocale().getLanguage().equals(desiredLanguage))
            chosenGreeter = greeter;
      }
      if (chosenGreeter == null)
         System.out.println("No suitable greeter.");
      else
         System.out.println(chosenGreeter.greet("Modular World"));
   }
}
```

The module declaration requires the service module and declares that the `GreeterService` is being used.

```
module v2ch09.useservice
{
   requires com.horstmann.greetsvc;
   uses com.horstmann.greetsvc.GreeterService;
}
```

As a result of the `provides` and `uses` declarations, the module that consumes the service is allowed access to the private implementation classes.

To build and run the program, first compile the service:

```
javac com.horstmann.greetsvc/module-info.java \
   com.horstmann.greetsvc/com/horstmann/greetsvc/GreeterService.java \
   com.horstmann.greetsvc/com/horstmann/greetsvc/internal/*.java
```

Then compile and run the consuming module:

```
javac -p com.horstmann.greetsvc \
   v2ch09.useservice/com/horstmann/hello/HelloWorld.java \
   v2ch09.useservice/module-info.java
java -p com.horstmann.greetsvc:v2ch09.useservice \
   -m v2ch09.useservice/com.horstmann.hello.HelloWorld
```

9.14 Tools for Working with Modules

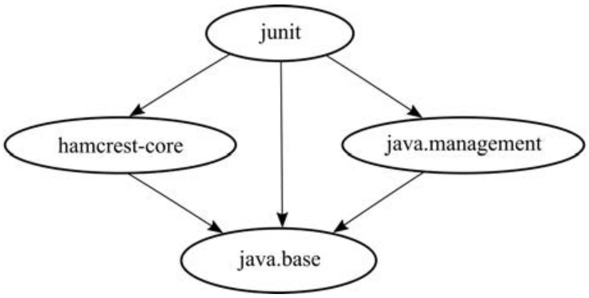
The `jdeps` tool analyzes the dependencies of a given set of JAR files. Suppose, for example, that you want to modularize JUnit 4. Run

```
jdeps -s junit-4.12.jar hamcrest-core-1.3.jar
```

The `-s` flag generates a summary output:

```
hamcrest-core-1.3.jar -> java.base
junit-4.12.jar -> hamcrest-core-1.3.jar
junit-4.12.jar -> java.base
junit-4.12.jar -> java.management
```

That tells you the module graph:



If you omit the `-s` flag, you get the module summary followed by a mapping from packages to required packages and modules. If you add the `-v` flag, the listing maps classes to required packages and modules.

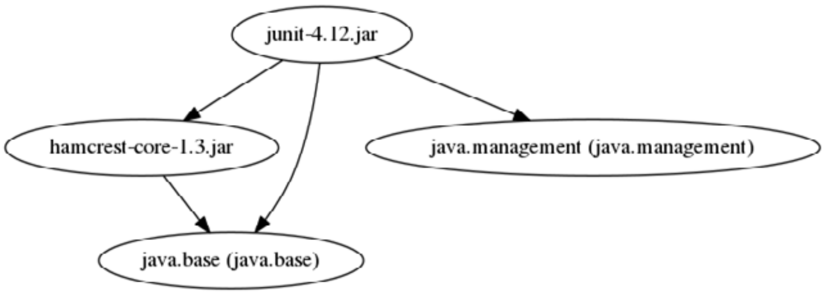
The `--generate-module-info` option produces `module-info` files for each analyzed module:

```
jdeps --generate-module-info /tmp/junit junit-4.12.jar hamcrest-core-1.3.jar
```

NOTE: There is also an option to generate graphical output in the "dot" language for describing graphs. Assuming you have the dot tool installed, run these commands:

```
jdeps -s -dotoutput /tmp/junit junit-4.12.jar hamcrest-core-1.3.jar
dot -Tpng /tmp/junit/summary.dot > /tmp/junit/summary.png
```

You get this `summary.png` image:



Use the jlink tool to produce an application that executes without a separate Java runtime. The resulting image is much smaller than the entire JDK. You specify the modules that you want to have included and an output directory.

```
jlink --module-path com.horstmann.greet.jar:v2ch09.exportedpkg.jar:$JAVA_HOME/jmods \
   --add-modules v2ch09.exportedpkg --output /tmp/hello
```

The output directory has a subdirectory bin with a java executable. If you run

```
bin/java -m v2ch09.exportedpkg
```

the main method of the module's main class is invoked.

The point of jlink is that it bundles up the minimal set of modules required to run the application. You can list them all:

```
bin/java --list-modules
```

In this example, the output is

```
v2ch09.exportedpkg
com.horstmann.greet
java.base@9
```

All modules are included in a *runtime image* file lib/modules. On my computer, that file is 23MB, whereas the runtime image of all JDK modules takes up 121MB. The entire application takes up 45MB, a fraction of the size of the JDK.

This can be the basis of a useful tool for packaging applications. You would still need to produce file sets for multiple platforms and launch scripts for the application.

NOTE: You can inspect the runtime image with the jimage command. However, the format is internal to the JVM, and runtime images are not meant to be generated or used by other tools.

Finally, the jmod tool builds and inspects the module files that are included with the JDK. When you look into the jmods directory inside the JDK, you will find a file with extension jmod for each module. There is no longer a rt.jar file.

Like JAR files, these files contain class files. In addition, they can hold native code libraries, commands, header files, configuration files, and legal notices. The JMOD files use the ZIP format. You can inspect their contents with any ZIP tool.

Unlike JAR files, JMOD files are only useful for linking—that is, for producing runtime images. There is no need for you to produce JMOD files unless you also want to bundle binary files such as native code libraries with your modules.

This brings us to the end of the chapter on the Java Platform Module System. The following chapter covers another important topic: security. Security has always been a core feature of the Java platform. As the world in which we live and compute gets more dangerous, a thorough understanding of Java security will be of increasing importance for many developers.

This page intentionally left blank

Index

Numbers

- (minus sign)
  in regular expressions, 144
  in URLs, 271
_ (underscore)
  in native method names, 787–788
  in SQL, 299, 329
  in URLs, 271
, (comma)
  decimal, 388, 395, 402
  in DTDs, 178–179
; (semicolon)
  in classpath, 302
  in decimal formatting, 400
  in method signatures, 807
  in SQL, 303
  not needed, in annotations, 468
: (colon)
  in classpath, 302
  in text files, 72
  in URLs, 260–261
!= operator (SQL), 299
? (question mark)
  in DTDs, 178–180
  in glob patterns, 127
  in prepared queries, 319
  in regular expressions, 144–145, 147
  in URLs, 270
/ (slash)
  in method signatures, 807
  in paths, 64, 115
  in URLs, 260
. (period)
  decimal, 388, 395, 402
  in method signatures, 807
  in regular expressions, 144–145, 156
  in URLs, 271
.., in paths, 116
^ (caret), in regular expressions, 144–147, 156
~ (tilde), in URLs, 271
' (apostrophe), entity reference for, 164
'. . .', in SQL, 299
". . .", in XML, 162
(. . .) (parentheses)
  in annotations, 477
  in method signatures, 807
  in regular expressions, 144–147, 151–154
[(array), type code, 96, 806
[. . .] (square brackets)
  in DOCTYPE declaration, 177
  in glob patterns, 127
  in regular expressions, 144–146
  in XPath, 195
{. . .} (curly brackets)
  in annotations, 478
  in glob patterns, 127
  in message formatting, 413–418
  in regular expressions, 144–147, 155
@ (at)
  in annotations, 468, 477
  in java command-line options, 522
  in URIs, 261
  in XPath, 195
¤ (currency sign), 401
$ (dollar sign), 403
  in native method names, 787
  in regular expressions, 144–145, 147, 155–157
€ (euro sign), 402, 419
* (asterisk)
  in DTDs, 178–180
  in glob patterns, 127
  in regular expressions, 144–147, 152
\ (backslash)
  in glob patterns, 127
  in paths, 64, 115
  in regular expressions, 144–145, 155, 157
\\|, in regular expressions, 73

& (ampersand)
  in CDATA sections, 165
  in entity references, 164
  parsing, 179
&&, in regular expressions, 146
&#, &#x, in character references, 164
# (number sign)
  in decimal formatting, 400
  in message formatting, 416–417
  in URLs, 260
% (percent sign)
  in locales, 395
  in SQL, 299, 329
  in URLs, 271
+ (plus sign)
  in DTDs, 178–179
  in regular expressions, 144–147
  in URLs, 271
  operator, identity for, 42
< (left angle bracket)
  in CDATA sections, 165
  in message formatting, 417
  parsing, 179
<!--. . .-->, <?. . .?>, <![CDATA[. . .]]>, in XML, 165
<. . .> (angle brackets), in regular expressions, 147
<> operator (SQL), 299
> (right angle bracket), in XML, 165
≤ operator, 417
= operator (SQL), 299
== operator, in enumerations, 105
| (vertical bar)
  in DTDs, 178–180
  in message formatting, 416
  in regular expressions, 144–146
  in text files, 72
\0, in regular expressions, 145
1931 CIE XYZ color specification, 734
2D graphics. *See* Java 2D API

A

\a, \A, in regular expressions, 145, 148
abort method (*LoginModule*), 562
absolute method (*ResultSet*), 333, 337
AbstractCellEditor class, 621–623
  isCellEditable method, 622
AbstractProcessor class, 488
AbstractTableModel class, 596
  getColumnName method, 597–598
  isCellEditable method, 619
accept method (ServerSocket), 244, 247–248
acceptChanges method (*CachedRowSet*), 340, 342
Accumulator functions, 42
Action listeners
  annotating, 469–475
  installing, 470
action.properties file, 461
ActionListener interface
  actionPerformed method, 452, 470
@ActionListenerFor annotation, 469, 485
ActionListenerInstaller class
  processAnnotations method, 470
add method
  of Area, 691–692
  of *AttributeSet*, 779, 783
  of DefaultMutableTreeNode, 633, 641
addActionListener method (ButtonFrame), 469–470
addAttribute method (AttributesImpl), 233
addBatch method (*Statement*), 355–356
addCellEditorListener method (*CellEditor*), 630
addColumn method (JTable), 608, 614
addRecipient method (MimeMessage), 288
addTreeModelListener method (*TreeModel*), 663, 671
addTreeSelectionListener method (JTree), 655
addURLs method (URLClassLoader), 537
Adleman, Leonard, 568, 587
AES (Advanced Encryption Standard), 579
  generating keys in, 580–585
aes/AESTest.java, 582
aes/Util.java, 583
Affine transformations, 706, 740
AffineTransform class, 706–709
  constructor, 707–708
  get*Xxx*Instance methods, 706–709, 711
  setTo*Xxx* methods, 707, 709
AffineTransformOp class, 740
  constructor, 747
  TYPE_*XXX* fields, 741, 747
afterLast method (*ResultSet*), 333, 337
Agent code, 499–500
Aliases, for namespaces, 185, 201
allMatch method (*Stream*), 17
allocate method (ByteBuffer), 138, 140
Alpha channel, 712–716

Alpha composites, 721
AlphaComposite class, 715
  getInstance method, 715, 721
&amp;, entity reference, 164
Anchor rectangles, 702
andFilter method (RowFilter), 607, 616
AnnotatedConstruct interface, 489
AnnotatedElement interface
  getAnnotation method, 470, 474, 488–489
  getAnnotations method, 474
  getAnnotationsByType method, 474, 488–489
  getDeclaredAnnotations method, 475
  isAnnotationPresent method, 474
Annotation interface
  extending, 475
  methods of, 476
Annotation interfaces, 469, 475–476
  predefined, 482–488
Annotation processors, 488–489
  at bytecode level, 470, 493
  at runtime, 470
Annotations, 467–475
  applicability of, 484–485
  documented, 485–486
  elements of, 468, 476
  evaluating, 478
  for compilation, 483
  for local variables, 480
  for packages, 479
  generating source code with, 490–493
  inherited, 487
  marker, 477
  meta, 469, 483–488
  never set to null, 478
  no annotations for, 481
  no circular dependencies in, 479
  processing tools for, 467
  repeatable, 488
  retaining, 485
  single value, 477
  source-level, 488–493
  standard, 482–488
  syntax of, 475–482
  transient, 486
  vs. Javadoc comments, 468
ANY element content (DTD), 178
anyMatch method (*Stream*), 17
Apache, 159
  Commons CSV library, 519–520
  Derby, 302
    connecting to, 305
    drivers for, 302–304
    populating, 315–318
    starting, 303
  DOM parser, 166
  Tomcat, 359
Apollo 11, launch of, 366, 375
&apos;, entity reference, 164
append method
  of *Appendable*, 62–63
  of Path2D, 681, 691, 711
Appendable interface, 61–63
appendChild method (*Node*), 213, 215
Applets
  executing, 562, 577
  security mechanisms in, 533
Application servers, 359
Applications
  business logic vs. visual representation of, 294–295
  client/server, 295
  configuring, 160–161
  enterprise, 358–360
  executing:
    loading classes for, 534
    without a separate Java runtime, 530
  localizing, 421–423
  monitoring, 499–500
  paid, 541
  server-side, 269–278
  signing, 577
  web, 358–360
applyPattern method (MessageFormat), 415
apply-templates element (XSLT), 225
Arc2D class, 674–675, 678
Arc2D.Double class, 675, 677, 690
Arc2D.Float class, 675, 677
ArcMaker class, 682
Arcs
  bounding rectangles of, 675, 678
  closure types of, 678
  computing angles of, 678
Area class
  add method, 691–692
  exclusiveOr method, 692
  intersect method, 692
  subtract method, 692
ARGB (Alpha, Red, Green, Blue), 716, 735

ARRAY data type (SQL), 357–358
ArrayIndexOutOfBoundsException, 820
Arrays
  converting to/from streams, 5, 26, 50
  getting from a database, 358
  in annotation elements, 478
  in native code, 816–819
  of primitive types, 818
  of strings, 154
  type code for, 96, 806
  type use annotations in, 480
Arrays class
  stream method, 5, 9, 44
ArrayStoreException, 820
asCharBuffer method (ByteBuffer), 138
ASCII encoding, 76, 155
  in property files, 423
  native code and, 795
ASM library, 493–501
asMatchPredicate method (Pattern), 149
ASP (Active Server Pages), 269
asPredicate method (Pattern), 149
Associative operations, 42
Atomic operations, 121
ATTLIST declaration (DTD), 180
attribute element (XML Schema), 187
Attribute interface, 776
  getCategory method, 778, 783
  getName method, 783
  implementing, 778
Attribute sets, 778
Attributes (XML)
  enumerating, 170
  in XML Schema, 187
  legal, 180
  names of, 162
  namespace of, 201
  values of, 162
    accessing in XPath, 195
    copying with XSLT, 226
    default (DTDs), 181
    normalizing, 181
  vs. elements, 163–164, 181, 223
Attributes interface
  get*Xxx* methods, 208
AttributeSet interface, 777
  add method, 779, 783
  get method, 779, 783
  remove method, 784
  toArray method, 784
AttributesImpl class
  addAttribute method, 233
  clear method, 233
atZone method (LocalDateTime), 375
auth/AuthTest.java, 551
auth/jaas.config, 552
Authentication, 549–562, 572–574
  role-based, 553
  through a trusted intermediary, 572–573
Autoboxing, 593
AutoCloseable interface
  close method, 62
Autocommit mode (databases), 353–355
Autoflushing, 69
Auxiliary files, generated, 467
available method (InputStream), 57–58
availableCharsets method (Charset), 77
average method (primitive streams), 44, 46–47
averaging*Xxx* methods (Collectors), 36
AWT (Abstract Window Toolkit), 671–721

B

B (byte), type code, 96, 806
\b, \B, in regular expressions, 148
Banding, 751
Banner class
  getPageCount method, 760
  layoutPages method, 760
Banners, printing, 759–769
BASE64Encoder class, 510
BaseStream interface
  iterator method, 29
  parallel method, 53
  unordered method, 53
BasicFileAttributes interface, 123
  methods of, 124
BasicStroke class, 693–701
  constructor, 701
Batch updates (databases), 354–357
BCP 47 memo, 390, 393
Bean info classes, generated, 467
beforeFirst method (*ResultSet*), 333, 337
between method (Duration), 363, 365
Bevel join, 694
Bézier curves, 680
Bicubic, bilinear interpolations, 741, 747

BIG_ENDIAN constant (ByteOrder), 138
Big5 encoding, 418, 421
BigDecimal class, 103
Big-endian order, 76, 79, 420
Binary data
  converting to Unicode code units, 68
  reading/writing, 78–79
  vs. text, 68
Bindings interface, 446
  get, put methods, 447
Birthdays, calculating, 367
BitSet interface, 43
BLOB data type (SQL), 301, 357
Blob interface, 326
  getBinaryStream method, 326–327
  getBytes method, 326–327
  length method, 327
  setBinaryStream method, 327
BLOBs (binary large objects), 326
  creating empty, 328
  placing in database, 326
Blocking
  by I/O methods, 57
  by network connections, 236, 240, 252–259
Blur filter, 742
BMP format, 722
body method (HttpResponse), 280, 286
BodyHandlers class
  discarding method, 281
  ofString method, 280–281
BodyPublishers class
  ofString method, 280
Book class, 759
book/Banner.java, 763
book/BookTestFrame.java, 761
book/PrintPreviewCanvas.java, 767
book/PrintPreviewDialog.java, 766
BOOLEAN data type (SQL), 301, 357
boolean type
  printing, 69
  streams of, 43–49
  type code for, 96, 806
  vs. C types, 793
  writing in binary format, 78
Bootstrap class loader, 535–536
Bounding rectangles, 675
boxed method (primitive streams), 44, 46–47
Bray, Tim, 161
Breadth-first enumerations, 649–650
breadthFirstEnumeration method (DefaultMutableTreeNode), 649, 654
Browsers
  forms in, 269–278
  response pages in, 269
Buffer class, 139–141
  capacity method, 141
  clear method, 140
  flip method, 140
  hasRemaining method, 137
  limit method, 137
  mark method, 140–141
  position method, 141
  remaining method, 140–141
  reset method, 140–141
  rewind method, 140–141
BufferedImage class, 702, 732
  constructor, 733, 738
  getColorModel method, 735, 738
  getRaster method, 733, 738
  TYPE_BYTE_GRAY field, 736, 738
  TYPE_BYTE_INDEXED field, 738
  TYPE_INT_ARGB field, 733–734, 738
BufferedImageOp interface, 732
  filter method, 740, 747
  implementing, 740
BufferedReader class
  lines method, 72
  readLine method, 71
Buffered*Xxx*Stream classes, 67
Buffers, 139–141
  capacity of, 139
  flushing, 57, 69
  in-memory, 62
  limits of, 139
  marks in, 139
  positions in, 132, 139
  traversing all bytes in, 132
  vs. random access, 131
BufferUnderflowException, 137
@BugReport annotation, 486
build method
  of HttpClient.Builder, 279–280, 285
  of HttpRequest.Builder, 286
Bundle classes, 424–426
Butt cap, 693
ButtonFrame class, 451
  addActionListener method, 469–470

buttons1/ButtonFrame.java, 455
buttons2/action.properties, 465
buttons2/ButtonFrame.java, 464
buttons3/ButtonFrame.java, 473
Byte order mark, 76, 420
byte type
  streams of, 43–49
  type code for, 96, 806
  vs. C types, 793
BYTE_ARRAY class (DocFlavor), 770
ByteArrayClass class, 459
ByteArrayClassLoader class, 460
ByteArrayOutputStream class, 110
ByteBuffer class, 132, 139–141
  allocate method, 138, 140
  asCharBuffer method, 138
  get, get*Xxx* methods, 133, 137–138
  order method, 133, 138
  put, put*Xxx* methods, 133, 137–138
  wrap method, 138, 140
bytecodeAnnotations/EntryLogger.java, 495
bytecodeAnnotations/EntryLoggingAgent.java, 500
Bytecodes
  engineering, 493–501
    at load time, 499–500
    with hex editor, 548
  verifying, 545–549
  writing to memory, 459–460
ByteLookupTable class, 742
  constructor, 748
ByteOrder class
  BIG_ENDIAN, LITTLE_ENDIAN constants, 138
Byte-oriented input/output streams, 56
Bytes, reading/writing, 56–59

C

C (char), type code, 96, 806
C programming language
  array types in, 816–819
  calling:
    from Java programs, 786–792
    Java methods from, 809–815
  database access in, 292
  embedding JVM into, 825–830
  FILE* type in, 59
  pointers in, 786
  strings in, 795
  types, vs. Java types, 793
\c, in regular expressions, 145
C++ programming language
  accessing JNI functions in, 796
  array types in, 816
  embedding JVM into, 825–830
  exceptions in, 820
  for native methods, 786, 789
  pointers in, 786, 811
Cached row sets, 339–344
CachedRowSet interface, 339–342
  acceptChanges method, 340, 342
  execute method, 340, 342
  getPageSize method, 340, 342
  getTableName method, 341–342
  nextPage method, 340, 342
  populate method, 339, 342
  previousPage method, 342
  setPageSize method, 340, 342
  setTableName method, 341–342
CachedRowSetImpl class, 510
Caesar cipher, 539–544
Calendar class, 361
  formatting objects of, 404
  weekends in, 368
call escape (SQL), 329
call method (*CompilationTask*), 458, 466
Callable interface, 458
Callback interface, 554
CallbackHandler interface
  handle method, 561
CallNonvirtual*Xxx*Method functions (C), 811, 815
CallStatic*Xxx*Method functions (C), 810, 815
Call*Xxx*Method functions (C), 809, 814–815
cancelCellEditing method (*CellEditor*), 622–623, 630
cancelRowUpdates method (*ResultSet*), 335, 338
canInsertImage method (ImageWriter), 725, 732
capacity method (Buffer), 141
Carriage return character, displaying, 170
Casts, type use annotations in, 480
catalog element (XML), 177
CatalogFeatures class
  defaults method, 178, 184
CatalogManager class
  catalogResolver method, 178, 184
Catalogs, 352
CDATA declaration (DTD), 180–181
CDATA sections (XML), 165
Cell editors (Swing), 619–623

Cell renderers (Swing)
  for tables, 600, 617–619
  for trees, 651–654
CellEditor interface
  add/removeCellEditorListener methods, 630
  cancelCellEditing method, 622–623, 630
  getCellEditorValue method, 620, 622–623, 630
  isCellEditable method, 630
  shouldSelectCell method, 622–623, 630
  stopCellEditing method, 622–623, 630
Cells (Swing)
  editing, 619–630
  selecting, 604
Certificates, 550, 569–571
  for software developers, 577
  Java Plug-in and, 577
  managing, 575–576
  publishing fingerprints of, 570
  signing, 574–576
CertificateSigner class, 575
CGI (Common Gateway Interface), 269
Chain of trust, 574
Channels, 253
  for files, 132
Channels class
  newInputStream method, 259
  newOutputStream method, 253, 259
char type
  streams of, 43–49
  type code for, 96, 806
  vs. C types, 793
CHAR_ARRAY class (DocFlavor), 770
Character classes, 144
CHARACTER data type (SQL), 301, 357
Character encodings, 68, 75–78
  character order in, 407
  explicitly specified, 77
  of source files, 420–421
  partial, 77
  platform, 77–78, 418
Character references (XML), 164
CharacterData interface
  getData method, 169, 175
Characters
  differences between, 408
  escaping, 73
  normalizing, 409
  outlines of, 710
  printing, 69
  writing in binary format, 78
characters method (ContentHandler), 203, 208
CharBuffer class, 62, 139
  get method, 138
  put method, 139
CharSequence interface, 62
  charAt method, 63
  chars method, 44
  codePoints method, 44, 48
  length method, 63
  splitting, 6
  subSequence method, 63
  toString method, 63
Charset class
  availableCharsets method, 77
  defaultCharset method, 77, 419
  forName method, 77
Checkboxes (Swing), 617
checked attribute (HTML, XML), 162
Checker framework, 480
checkError method (PrintWriter), 69–70
Child elements (XML), 163
  namespace of, 200
Child nodes (Swing), 630
  adding, 633
  connecting lines for, 636–637
children method (*TreeNode*), 649
choice element (XML Schema), 187
choice keyword (message formatting), 416
Church, Alonzo, 366
Cipher class, 579–580
  doFinal method, 580, 582, 585–586
  getInstance method, 579, 584
  get*Xxx*Size methods, 584
  init method, 584
  update method, 579, 582, 585–586
  *XXX*_MODE modes, 579
CipherInputStream class
  read method, 586
CipherOutputStream class, 585
  constructor, 586
  flush method, 586
  write method, 586
Ciphers
  Caesar, 539–544
  keys for:
    generating, 580–585
    public, 566–573, 587–590

performance of, 587
streams for, 585–586
symmetric, 579–580, 587
Class class
forName method, 460
getClassLoader method, 535, 544
getFields method, 665
getResourceAsStream method, 517
implementing *AnnotatedElement*, 470
.class file extension, 534
Class files, 534
corrupted, 546–548
encrypted, 539–545
format of, 493
loading, 534–535
modifying, 493–499
portability of, 420
transformers for, 500
verifying, 545–549
Class loaders, 461, 534–549
as namespaces, 538–539
bootstrap, 535–536
context, 537
extension, 535
hierarchy of, 536–538
separate for each web page, 539
specifying, 537
system, 535
writing, 539–545
Class path, adding JAR files to, 537
Class references, in native code, 803
Classes
adding validation to, 101
annotating, 468, 479, 483
compiling on the fly, 459
encapsulation of, 504
externalizable, 96
inheritance trees of, 650
loading, 460
nonserializable, 101
platform, 535
resolving, 534
separate for each web page, 539
serializable, 89, 109
deserializing, 113–114
static inner, 92
versioning, 107–110
Classifier functions, 35
ClassLoader class, 535
defineClass method, 539, 544–545
extending, 460, 539
findClass method, 539, 544
getParent method, 544
getSystemClassLoader method, 544
loadClass method, 537, 539
Classloader inversion, 537
classLoader/Caesar.java, 543
classLoader/ClassLoaderTest.java, 541
CLASSPATH environment variable, 535
CLEAR composition rule, 714
clear method
of AttributesImpl, 233
of Buffer, 140
clearParameters method (*PreparedStatement*), 325
client/HttpClientTest.java, 282
Client/server applications, 295
Clients
connecting to servers, 238–240
multiple, serving, 247–251
CLinker interface, 847
clip method (Graphics2D), 672, 710–712, 752
Clipping shapes, 672, 710–712
printing, 752
setting region for, 672
CLOB data type (SQL), 301, 357
Clob interface, 326
getCharacterStream method, 326–327
getSubString method, 326–327
length method, 327
setCharacterStream method, 327
CLOBs (character large objects), 326
creating empty, 328
placing in database, 326
clone method (Object), 89, 110
Cloning, 110–113
close method
of *AutoCloseable*, 62
of *Closeable*, 61–63
of *Connection*, 309, 312, 360
of FileLock, 143
of *Flushable*, 61
of InputStream, 57–58
of OutputStream, 59
of *ResultSet*, 311–312
of ServerSocket, 247
of *Statement*, 310, 312
of *XMLStreamWriter*, 222

Closeable interface, 61
  close method, 61–63
  flush method, 61
closeEntry method (Zip*Xxx*Stream), 85–87
closeOnCompletion method (*Statement*), 310, 312
closePath method (Path2D), 680, 691
Closure types, 678
cmd shell, 419
Code generation, annotated, 467–475, 484
Code points, 11
Code units, 44
  in regular expressions, 145
  writing, 82
Codebreakers, The (Kahn), 540
codePoints method (*CharSequence*), 44, 48
Collation, 407–413
collation/CollationTest2.java, 410
CollationKey class
  compareTo method, 413
Collator class, 408
  compare method, 412
  equals method, 412
  get/setDecomposition methods, 413
  get/setStrength methods, 412
  getAvailableLocales method, 412
  getCollationKey method, 410, 413
  getInstance method, 412
collect method (*Stream*), 26–29, 43
collecting/CollectingIntoMaps.java, 32
collecting/CollectingResults.java, 27
collecting/DownstreamCollectors.java, 38
collectingAndThen method (Collectors), 37, 40
Collection interface
  parallelStream method, 2–4, 49–53
  stream method, 2–4
Collections
  iterating over elements of, 2–4
  vs. streams, 3
Collections class
  sort method, 408
Collector interface, 26
Collectors, 26–41
  composing, 38
  downstream, 36–41, 50
Collectors class
  averaging*Xxx* methods, 36
  collectingAndThen method, 37, 40
  counting method, 36, 40
  filtering method, 38, 41
  flatMapping method, 37, 41
  groupingBy method, 35–40
  groupingByConcurrent method, 35–36, 50
  joining method, 27, 30
  mapping method, 37, 41
  maxBy, minBy methods, 37, 40
  partitioningBy method, 35–36, 38
  reducing method, 38
  summarizing*Xxx* methods, 27, 30, 37
  summing*Xxx* methods, 36, 40
  teeing method, 38
  toCollection method, 27, 30
  toConcurrentMap method, 32, 34
  toList method, 26, 30
  toMap method, 31–34
  toSet method, 26, 30, 36
  toUnmodifiableList method, 30
  toUnmodifiableMap method, 34
  toUnmodifiableSet method, 30
Color choosers, 621
Color class, 701
  constructor, 740
  getRGB method, 740
  translating values into pixel data, 736
Color space conversions, 742
ColorConvertOp class, 740, 742
ColorModel class, 736
  getDataElements method, 739
  getRGB method, 735, 739
Colors
  components of, 712
  composing, 713–716
  interpolating, 701–702
  negating, 741
  solid, 672
Columns (databases)
  accessing by number, in result set, 309
  names of, 295
  number of, 344
Columns (Swing)
  accessing, 601–602
  adding, 608
  detached, 593
  hiding, 608
  names of, 597–598
  rendering, 600
  resizing, 593–594, 602–603
  selecting, 604
com.sun.security.auth.module package, 550–551

Combo box editors, 620
Comments (XML), 165
commit method
  of *Connection*, 354–356
  of *LoginModule*, 562
commonPool method (ForkJoinPool), 51
Commons CSV library, 519–520
Comparable interface, 15
  compareTo method, 606
Comparator interface, 15, 408
Comparators, 606
compare method (Collator), 412
compareTo method
  of CollationKey, 413
  of *Comparable*, 606
  of Instant, 363
  of String, 407
Compilable interface, 450–451
  compile method, 451
CompilationTask interface, 457
  call method, 458, 466
compile method
  of Compilable, 451
  of Pattern, 149, 155–156
CompiledScript interface
  eval method, 451
Compiler
  annotations for, 483
  invoking, 457
  just-in-time, 825
compiler/CompilerTest.java, 462
Complex types, 185
complexType element (XML Schema), 186
Composite interface, 715
composite/CompositeComponent.java, 718
composite/CompositeTestFrame.java, 717
composite/Rule.java, 720
Composition rules, 672–673, 712–721
Computer Graphics: Principles and Practice (Hughes et al.), 680, 734
concat method (*Stream*), 14
Confidential information, 578
Configuration files, 114, 141
connect method
  of Socket, 241
  of URLConnection, 262, 264, 268
Connection interface
  close method, 309, 312, 360
  commit method, 354–356
  createBlob, createClob methods, 326, 328
  createStatement method, 308–309, 331–332, 334, 336, 354
  getAutoCommit method, 355–356
  getMetaData method, 343, 352
  getWarnings method, 314
  prepareStatement method, 319–320, 325, 331, 336
  releaseSavepoint method, 354, 356
  rollback method, 354–356
  setAutoCommit method, 353, 355–356
  setSavepoint method, 356
Connections (databases)
  closing, 312
    using row sets after, 339
  debugging, 288
  pooling, 359
  starting new threads, 248
console method (System), 419
Constructive area geometry, 691
Constructor class, 470
Constructors
  annotating, 479
  bypassing, 113–114
  invoking from native code, 811
  no-argument, 101, 113
  type use annotations in, 480
Content types, 262
ContentHandler class, 203–204
  characters method, 203, 208
  startDocument, endDocument methods, 203, 207
  startElement, endElement methods, 203–208
Context class loader, 537
Control points
  dragging, 682
  of curves, 679–680
  of shapes, 681
convert*Xxx*IndexToModel methods (JTable), 604, 614
Convolution operation, 742
ConvolveOp class, 740, 742–743
  constructor, 748
CookieHandler class
  setDefault method, 274
Cookies, 274
Coordinate system
  custom, 673
  translating, 752
Coordinate transformations, 703–709

Copies class, 776–779
  getValue method, 779
CopiesSupported class, 776
copy method (Files), 120–122, 130
CORBA (Common Object Request Broker Architecture), 504, 535
Core Swing (Topley), 592, 630, 643
count function (XPath), 195
count method (*Stream*), 3–4, 16
counting method (Collectors), 36, 40
Country codes, 35, 390
CRC32 checksum, 88, 131, 133
CRC32 class, 133
CREATE TABLE statement (SQL), 300
  executing, 308, 310, 326
  in batch updates, 355
createBindings method (*ScriptEngine*), 447
createBlob, createClob methods (*Connection*), 326, 328
createCachedRowSet method (*RowSetFactory*), 339–340, 342
createDirectory, createDirectories methods (Files), 119–120
createElement, createElementNS methods (Document), 213, 215
createFile method (Files), 119–120
createFilteredRowSet method (*RowSetFactory*), 342
createImage*Xxx*Stream methods (ImageIO), 724, 730
createJdbcRowSet, createJoinRowSet methods (*RowSetFactory*), 342
createLSOutput method (*DOMImplementationLS*), 215
createLSSerializer method (*DOMImplementationLS*), 214
createPrintJob method (*PrintService*), 771–772
createStatement method (*Connection*), 308–309, 331–332, 334, 336, 354
createTemp*Xxx* methods (Files), 120
createTextNode method (*Document*), 213, 215
createWebRowSet method (*RowSetFactory*), 342
createXMLStreamReader method (XMLInputFactory), 211
createXMLStreamWriter method (XMLOutputFactory), 217, 221
creationTime method (*BasicFileAttributes*), 124
Credit card numbers, 578
crypt program, 582
Cryptography and Network Security (Stallings), 564
CSV files, 519–520
Cubic curves, 679–680
CubicCurve2D class, 674, 680
CubicCurve2D.Double class, 675, 677, 690
CubicCurve2D.Float class, 675, 677
Currencies, 402–403
  available, 403
  formatting, 401
  identifiers for, 402
Currency class, 402–403
  getAvailableCurrencies method, 403
  getCurrencyCode method, 403
  getDefaultFractionDigits method, 403
  getInstance method, 402–403
  getNumeric*Xxx* methods, 403
  getSymbol method, 403
  toString method, 403
curveTo method (Path2D.Float), 680, 691
Cyclic references, 113
Cygwin, 790
  compiling invocation API, 830
  OpenSSL in, 576

D

D (double), type code, 96, 806
d literal (SQL), 328
\d, \D, in regular expressions, 146
Dashed lines, 694–695
Data
  encrypting/decrypting, 585
  fingerprints of, 563–566
  signed, 566–569
Data sources (for JNDI service), 359
Data types
  codes for, 96, 806
  in Java vs. C, 793
  mangling names of, 806
  print services for, 769–770
database.properties file, 315, 358
DatabaseMetaData interface, 343–353
  getJDBC*Xxx*Version methods, 352
  getMaxConnection method, 352
  getMaxStatements method, 311, 352
  getSQLStateType method, 312
  getTables method, 343, 352
  supportsBatchUpdates method, 355, 357
  supportsResultSet*Xxx* methods, 332, 338

Databases, 291–360
  accessing, in C language, 292
  autocommit mode of, 353–355
  batch updates for, 354–357
  caching prepared statements, 320
  changing data with SQL, 300
  connections to, 302, 304–307, 316
    closing, 312, 316
    in web and enterprise applications, 358–360
    pooling, 359
  drivers for, 293–294
  duplicating data in, 297
  error handling in, 355
  escape syntax in, 328–329
  integrity of, 353
  JAR files for, 302
  keys in, 330–331
  LOBs in, 326–328
  metadata for, 343–353
  modifying, 339
  native storage for XML in, 358
  numbering columns in, 309
  outer joins in, 328
  populating, 315–318
  registering classes for, 304
  saving objects to, 487
  scalar functions in, 328
  schemas for, 352
  scrollable/updatable result sets in, 332
  setting up parameters in, 340
  starting, 303
  stored procedures in, 329
  structure of, 295, 343
  synchronization of, 341
  tools for, 345
  truncated data from, 313
  URLs of, 302
DataFlavor class, 505
DataInput interface
  readBoolean method, 79–80
  readChar method, 79–81
  readDouble method, 79–80, 90, 102
  readFloat method, 79–80
  readFully method, 80
  readInt method, 79–81, 90
  readLong method, 79–80
  readShort method, 79–80
  readUTF method, 79–80
  skipBytes method, 80
DataInputStream class, 60, 65, 79
DataIO class
  *xxx*FixedString methods, 82
DataOutput interface, 78
  writeBoolean method, 78, 80
  writeByte method, 78, 80
  writeChar method, 78, 80–81
  writeChars method, 78–80
  writeDouble method, 78, 80, 90, 102
  writeFloat method, 78, 80
  writeInt method, 78, 80–81, 90
  writeLong method, 78, 80
  writeShort method, 78, 80
  writeUTF method, 78–80
DataOutputStream class, 60, 79
DataSource interface, 359
DataTruncation class, 313
  get*Xxx* methods, 314
Date and Time API, 361–385
  legacy code and, 384–385
Date class (java.sql), 384
  valueOf method, 385
Date class (java.util), 96, 103, 361, 384
  formatting objects of, 404
  months and years in, 367
  toInstant method, 384–385
DATE data type (SQL), 301, 328, 357
dateFilter method (RowFilter), 607, 616
DateFormat class, 381, 385, 405
dateFormat/DateTimeFormatTest2.java, 406
DateFormatter class (java.text), 404
Dates
  computing, 367, 372–373
  filtering, 607
  formatting, 379–384, 388–389, 403–407, 413–415
  literals for, 328
  local, 366–371
  nonexistent, 404
  parsing, 381
datesUntil method (LocalDate), 368, 371
DateTimeFormatter class, 379–384, 403–407
  format method, 379, 383, 407
  legacy classes and, 384–385
  ofLocalized*Xxx* methods, 380, 383, 403, 407
  ofPattern method, 381, 384
  parse method, 381

toFormat method, 381, 385
withLocale method, 380, 384, 404, 407
DateTimeParseException, 407
DateTimeSyntax class, 779
Daylight savings time, 375–379
DayOfWeek enumeration, 368
getDisplayName method, 381, 405
dayOfWeekInMonth method (TemporalAdjusters), 373
Days of week, 405
DBeaver program, 345
DDL statement (SQL), 310, 326
Debugging
in JNI, 826
JDBC-related problems, 306
locales, 394
mail connections, 288
streams, 15
with telnet, 235–238
DECIMAL data type (SQL), 301, 357
Decimal separators, 388, 395, 402
DecimalFormat class, 399–402
DecimalFormatSymbols class, 401
Declaration annotations, 479–481
decode method (URLDecoder), 278
Decomposition, 409
default statement, 475
DefaultCellEditor class, 645
constructor, 629
variations of, 619
defaultCharset method (Charset), 77, 419
DefaultHandler class, 204
DefaultMutableTreeNode class, 632, 649–651, 653
add method, 633, 641
constructor, 641
pathFromAncestorEnumeration method, 650
setAllowsChildren method, 639, 641
*xxx*FirstEnumeration methods, 649, 654
*xxx*OrderEnumeration methods, 650, 654
defaultPage method (PrinterJob), 758
DefaultRowSorter class
setComparator method, 606, 615
setRowFilter method, 606–608, 615
setSortable method, 605, 615
defaults method (CatalogFeatures), 178, 184
DefaultTableCellRenderer class, 618
DefaultTableModel class
isCellEditable method, 619
DefaultTreeCellRenderer class, 651–654
getTreeCellRendererComponent method, 652–653
set*Xxx*Icon methods, 652, 654
DefaultTreeModel class, 632, 642, 664
automatic notifications by, 643
getPathToRoot method, 644
insertNodeInto method, 643, 649
isLeaf method, 639
nodeChanged method, 643, 649
nodesChanged method, 649
reload method, 643, 649
removeNodeFromParent method, 643, 649
setAsksAllowsChildren method, 639, 641
defaultWriteObject method (ObjectOutputStream), 102
defineClass method (ClassLoader), 539, 544–545
delete method (Files), 121–122
DELETE method (HttpRequest.Builder), 286
DELETE statement (SQL), 300
executing, 308, 310, 326
in batch updates, 355
vs. methods of *ResultSet*, 335
DeleteGlobalRef function (C), 803
deleteIfExists method (Files), 121–122
deleteRow method (*ResultSet*), 335, 337
Delimiters, in text files, 72
@Deprecated annotation, 483
Depth-first enumerations, 649–650
depthFirstEnumeration method (DefaultMutableTreeNode), 649, 654
Derby databases. *See* Apache, Derby
derbyclient.jar file, 302
DES (Data Encryption Standard), 579
Deserialization, 113–114
DestroyJavaVM function (C), 826, 830
Device coordinates, 704
Diagnostic interface, 458
get*Xxx* methods, 466
DiagnosticCollector class, 458
constructor, 466
getDiagnostics method, 466
DiagnosticListener interface, 458
DialogCallbackHandler class, 554
Dictionary ordering, 408
digest method (MessageDigest), 565–566
DigiCert, 570, 574

Digital fingerprints, 95–100, 563–566
  computing, 108
  different for a class and its objects, 98
Digital signatures, 562–577
  verifying, 569–571
Direct buffers, 818
Directories
  creating, 119–120
  current, 129
  hierarchical structure of, 630
  printing all subdirectories of, 128
  traversing, 124–129
  user's working, 64
DirectoryStream interface, 126
discarding method (BodyHandlers), 281
distinct method (*Stream*), 15–16, 50
dividedBy method (Duration), 366
Doc interface, 771
DocAttribute interface, 776
  implementing, 778
  printing attributes of, 780–783
DocAttributeSet interface, 777–778
DocFlavor class, 769–770, 772
DocPrintJob interface
  getAttributes method, 784
  print method, 772
DOCTYPE declaration (DTD), 177
  including in output, 214
Document interface, 166
  create*Xxx* methods, 213, 215
  getDocumentElement method, 166, 174
Document flavors, for print services, 769–770
DocumentBuilder class
  newDocument method, 213, 215, 228
  parse method, 173
  setEntityResolver method, 178, 183
  setErrorHandler method, 183
DocumentBuilderFactory class
  isIgnoringElementContentWhitespace method, 185
  isNamespaceAware method, 202
  isValidating method, 185
  newDocumentBuilder method, 166, 173, 213
  newInstance method, 166, 173, 201
  newNSInstance method, 201–202, 205
  setIgnoringElementContentWhitespace method, 182, 185
  setNamespaceAware method, 188, 201–202, 205, 213
  setValidating method, 182, 185
@Documented annotation, 483, 485–486
doFinal method (Cipher), 580, 582, 585–586
DOM (Document Object Model) parser, 165–166, 203
  namespace-awareness of, 201, 205
  trees in:
    accessing with XPath, 194–199
    analyzing, 168–170
    building, 203, 212–223, 228
    writing, 214–216
dom/JSONConverter.java, 170
DOMImplementationLS interface
  createLSOutput method, 215
  createLSSerializer method, 214
DOMResult class, 228, 233
DOMSource class, 216, 227
DOUBLE data type (SQL), 301, 357
double type
  printing, 69
  streams of, 43–49
  type code for, 96, 806
  vs. C types, 793
  writing in binary format, 78
DoubleBuffer class, 139
doubles method
  of Random, 45
  of RandomGenerator, 48
  of SplittableRandom, 52
DoubleStream interface, 43–49
  average method, 44, 47
  boxed method, 44, 47
  mapToDouble method, 44
  max, min methods, 44, 47
  of method, 47
  sum, summaryStatistics methods, 44, 47
  toArray method, 44, 47
DoubleSummaryStatistics class, 27, 30–31, 44, 49
doubleValue method (Number), 396
Downstream collectors, 36–41, 50
draw method (Graphics2D), 673–675, 692–693
Drawings
  creating, 671–721
  printing, 749–759
draw*Xxx* methods (Graphics), 674

DriverManager class, 304
  getConnection method, 305, 307, 316, 360
  setLogWriter method, 306
DROP TABLE statement (SQL), 305
  executing, 308, 310
  in batch updates, 355
dropWhile method (*Stream*), 14
DSA (Digital Signature Algorithm), 567–568
DST, DST_*Xxx* composition rules, 714
DTDHandler class, 204
DTDs (Document Type Definitions), 176–185
  element content in, 178–179
  entities in, 181
  external, 177
  in XML documents, 162, 176–185
  locating, 177–178
  unambiguous, 180
  URLs for, 177
Duration class
  between method, 363, 365
  dividedBy method, 366
  getSeconds method, 363
  immutability of, 363
  isNegative, isZero methods, 366
  multipliedBy method, 366
  negated method, 366
  of*Xxx* methods, 365
  to*Xxx* methods, 363, 365–366
Dynamic links, 826
Dynamic web pages, 461–467

E

\e, \E, in regular expressions, 145
Echo servers, 246–247
Eclipse
  IDE, 513
  Yasson framework, 518
Edge detection, 743
element element (XML Schema), 186
ELEMENT element content (DTD), 178–179
Element interface, 166, 489
  getAttribute method, 170, 174
  getSimpleName method, 489
  getTagName method, 167, 174, 202
  setAttribute, setAttributeNS methods, 213–215
Elements (XML)
  child, 163
    accessing in XPath, 195
    namespace of, 200
  constructing, 213
  counting, in XPath, 195
  empty, 162
  legal attributes of, 180
  mixed content in, 163, 179
  names of, 167, 202
  root, 163, 185
  trimming whitespace in, 169
  vs. attributes, 163–164, 181, 223
Ellipse2D class, 674–675
Ellipse2D.Double, Ellipse2D.Float classes, 675, 677
Ellipses, 675
E-mails
  sending, 287–290
  terminating lines in, 287
employee/Employee.c, 805
employee/Employee.java, 804
employee/EmployeeTest.java, 804
EMPTY element content (DTD), 178
empty method
  of Optional, 21–22
  of *Stream*, 5, 9
Empty tags (XML), 162
Encapsulation, 504
  compile-time, 522
encode method (URLEncoder), 278
Encodings. *See* Character encodings
Encryption, 578–590
  decryption keys for, 540
  exporting strong methods of, 540
  final block padding in, 580
  of class files, 539–545
end method
  of Matcher, 157
  of *MatchResult*, 158
End cap styles, 693–695
End points, 679
End tags (XML), 162
endDocument method (ContentHandler), 203, 207
endElement method (ContentHandler), 203–208
End-of-line character. *See* Line feed
Enterprise applications, 358–360
Enterprise JavaBeans (EJBs), 295
Entity references (XML), 164, 181

Entity resolvers, 166, 177
ENTITY, ENTITIES attribute types (DTDs), 180–181
EntityResolver interface, 184, 204
  resolveEntity method, 177, 183
entries method (ZipFile), 88
Entrust, 574
Entry class, 607
  get*Xxx* methods, 616–617
EntryLogger class, 500
EntryLoggingAgent.mf file, 499
enum keyword, 104
enumeration element (XML Schema), 186
Enumeration interface, 88
  hasMoreElements method, 832–835
  nextElement method, 649, 832–835
Enumerations
  customizing serialization of, 104
  ignoring serialVersionUID fields, 110
  of nodes, in a tree, 649–651
  typesafe, 104–107
  using attributes for, 181
EnumSyntax class, 779
EOFException, 820
Epoch, 103, 362
equals method
  of *Annotation*, 476
  of Collator, 412
  of Instant, 363
error method (ErrorHandler), 183–184
Error handlers
  in native code, 819–825
  installing, 183
ErrorHandler class, 204
  error, fatalError methods, 183–184
  warning method, 183–184
Errors
  OutOfMemoryError, 821
  UnsatisfiedLinkError, 787
Escape hatch mechanism, 641
escape keyword (SQL), 329
Escapes
  in regular expressions, 73
  in SQL, 328–329
Essential XML (Box et al.), 159, 224
Euro symbol, 402, 419
eval method
  of CompiledScript, 451
  of *ScriptEngine*, 445–447
evaluate, evaluateExpression methods (*XPath*), 195–196, 199
Event handlers, annotating, 469–475
Event listeners, 468
EventHandler class, 471
EventListenerList class, 664
Evins, Jim, 600
evn pointer (C), 796
Exceptions
  ArrayIndexOutOfBoundsException, 820
  ArrayStoreException, 820
  BufferUnderflowException, 137
  DateTimeParseException, 407
  EOFException, 820
  FileNotFoundException, 274
  from native code, 819–825
  IllegalArgumentException, 211, 821
  IllegalStateException, 31, 724
  in C++, 820
  in SQL, 312–314
  IndexOutOfBoundsException, 724, 731
  InvalidObjectException, 110
  InvalidPathException, 115
  IOException, 62, 239
  MissingResourceException, 422
  NoSuchAlgorithmException, 566, 584
  NoSuchElementException, 835
  NotSerializableException, 102
  NullPointerException, 821
  OverlappingFileLockException, 142
  ParseException, 396, 398
  ReadOnlyBufferException, 132
  SocketTimeoutException, 241, 268
  SQLException, 312–314, 332, 354, 356
  SQLWarning, 332
  SyncProviderException, 341–342
  type use annotations in, 480
  UnknownHostException, 239
Exception*Xxx* functions (C), 820–821, 825
Exclusive lock, 142
exclusiveOr method (Area), 692
exec method (Runtime), 114
exec/ExecSQL.java, 316
ExecutableElement interface, 489
execute method
  of *CachedRowSet*, 340, 342
  of *RowSet*, 341
  of *Statement*, 310, 316, 329, 331
executeBatch method (*Statement*), 355–356

executeLargeBatch method (*Statement*), 356
executeLargeUpdate method (*Statement*), 310
executeQuery method
  of *PreparedStatement*, 320, 326
  of *Statement*, 308, 310, 332, 334
executeUpdate method
  of *PreparedStatement*, 320, 326
  of *Statement*, 308, 310, 331, 354
executor method (HttpClient.Builder), 285
ExecutorService interface, 458
exists method (Files), 123–124
EXIT statement (SQL), 303
exports keyword, 510, 512–513, 525
Extension class loader, 535
extern "C", in native methods (C++), 789
External entities, 181
Externalizable interface
  methods of, 103–104

F

F (float), type code, 96, 806
\f, in regular expressions, 145
Factoring algorithms, 568
fatalError method (ErrorHandler), 183–184
Field class
  getName, getType methods, 665
  implementing *AnnotatedElement*, 470
Fields
  accessing from native code, 801–806
  annotating, 468, 483
  transient, 101–102
File class
  separator constant, 64
  toPath method, 117–118
File pointers, 81
File systems, POSIX-compliant, 123
file: (URI scheme), 260
file.encoding property, 77
FileChannel class
  lock method, 141–143
  map method, 132, 137
  open method, 132, 137
  tryLock method, 141–143
FileHandler class, 420
FileInputStream class, 64–67
  constructor, 66
  getChannel method, 136
  read method, 56
fileKey method (*BasicFileAttributes*), 124
FileLock class
  close method, 143
  isShared method, 142
FileNotFoundException, 274
FileOutputStream class, 64–67
  constructor, 67
  getChannel method, 136
Files
  channels for, 132
  closing, 124, 126
  configuration, 141
  copying, 120–122
  creating, 119–120
  deleting, 121–122
  encrypted data in, 585
  filtering, 126–127, 723
  generated automatically, 467, 488
  hierarchical structure of, 630
  I/O modes of, 85
  locking, 141–143
  memory-mapped, 51, 131–141
  MIME type of, 118
  missing, 458
  moving, 120–122
  random-access, 81–85, 131
  reading/writing, 65, 118–119
    by one byte, 56–59
  total number of bytes in, 81
  traversing, 126–130
  with multiple images, 724–732
Files class, 115, 118–129
  copy method, 120–122, 130
  create*Xxx* methods, 119–120
  delete, deleteIfExists methods, 121–122
  exists method, 123–124
  find method, 125
  getOwner method, 123
  is*Xxx* methods, 123–124
  lines method, 6, 10, 51, 118
  list method, 124–125
  mismatch method, 118–119
  move method, 120–122
  newBuffered*Xxx* methods, 118–119
  newDirectoryStream method, 126, 129
  new*Xxx*Stream methods, 118–119
  probeContentType method, 118–119
  readAll*Xxx* methods, 118–119
  readAttributes method, 123–124
  readString method, 118–119

size method, 123–124
walk method, 125–126
walkFileTree method, 127–129
write, writeString methods, 118–119
FileSystem class
getPath method, 130–131
FileSystems class
newFileSystem method, 130–131
FileTime class
toInstant method, 385
FileTypeDetector class, 118
FileVisitor interface, 128–129
methods of, 127
fill method (Graphics2D), 673–674, 692
Filling shapes, 673
fill*Xxx* methods (Graphics), 674
filter method
of *BufferedImageOp*, 740, 747
of Optional, 19–20
of *Stream*, 3–11, 16
FilteredRowSet interface, 339
filtering method (Collectors), 38, 41
Filters
for images, 740–749
for numbers, 606–607
for table rows, 606–608
glob patterns for, 126–127
implementing, 607
Filter*Xxx*Stream classes, 65
Final block padding, 580
find method
of Files, 125
of Matcher, 157
findAll method (Scanner), 158
findAny method (*Stream*), 17
FindClass function (C), 802, 805, 810
findClass method (ClassLoader), 539, 544
findColumn method (*ResultSet*), 311
findFirst method (*Stream*), 16–17
Fingerprints. *See* Digital fingerprints
first method (*ResultSet*), 333, 337
firstDayOf*Xxx* methods (TemporalAdjuster), 373
firstValue method (HttpHeaders), 281, 286
#FIXED attribute (DTD), 181
flatMap method
of Optional, 22–26
of *Stream*, 12
flatMapping method (Collectors), 37, 41
flip method (Buffer), 140
FLOAT data type (SQL), 301, 357
float type
printing, 69
streams of, 43–49
type code for, 96, 806
vs. C types, 793
writing in binary format, 78
FloatBuffer class, 139
Floating-point numbers, 388, 395–403
flush method
of CipherOutputStream, 586
of *Closeable*, 61
of *Flushable*, 61, 63
of OutputStream, 57, 59
Flushable interface, 61–62
close method, 61
flush method, 61, 63
fn keyword (SQL), 328
Folders, icons for, 638–639, 651
followRedirects method (HttpClient.Builder), 279, 285
Font render context, 710
forEach method (*Stream*), 26, 29
forEachOrdered method (*Stream*), 26
Foreign functions, 846–847
Forest (Swing), 630, 638
ForkJoinPool class
commonPool method, 51
forLanguageTag method (Locale), 391, 394
Format class, 384, 415
format method
of DateTimeFormatter, 379, 383, 407
of Format, 415
of LocalDate, 404
of LocalDateTime, 404
of LocalTime, 404
of MessageFormat, 413–415
of NumberFormat, 396, 398
of String, 393
of ZonedDateTime, 404
Formatting
dates, 388–389, 403–407, 413–415
messages, 413–418
numbers, 388–389, 395–403, 413–415
formatting/Formatting.java, 382
Forms, processing, 269–278
forName method
of Charset, 77
of Class, 460

ForwardingJavaFileManager class
constructor, 467
getFileForOutput method, 467
fprintf function (C), 809
Frame class, 461
from method
of Instant, 384–385
of ZonedDateTime, 384–385
FROM statement (SQL), 298
FTP (File Transfer Protocol), 264
ftp: (URI scheme), 260, 264
Function interface
identity method, 31
@FunctionalInterface annotation, 483

G

\G, in regular expressions, 148
Gadget chains, 114
Garbage collection
arrays and, 818
native methods and, 797
GB encoding, 421
GeneralPath class, 674–675, 680
constructor, 690
generate method (*Stream*), 5, 9, 44
@Generated annotation, 483–484
generateKey method (KeyGenerator), 580, 585
Generators, converting to streams, 50
Generic types, type use annotations in, 480
get method
of *AttributeSet*, 779, 783
of *Bindings*, 447
of ByteBuffer, 133, 137
of CharBuffer, 138
of Optional, 20–23
of Paths, 115, 117
of *ScriptEngine*, 447
of ScriptEngineManager, 447
of *Supplier*, 11
GET method (HttpRequest.Builder), 279, 286
GET request (HTML), 270, 272
building, 279
getAddress method (InetAddress), 242–243
getAdvance method (TextLayout), 712
getAllByName method (InetAddress), 242–243
getAllowsChildren method (*TreeNode*), 640
getAnnotation method (*AnnotatedElement*), 470, 474, 488–489
getAnnotations method (*AnnotatedElement*), 474
getAnnotationsByType method (*AnnotatedElement*), 474, 488–489
GetArrayLength function (C), 816, 819
getAscent method (TextLayout), 712
getAs*Xxx* methods (Optional*Xxx*), 44, 48
getAttribute method (*Element*), 170, 174
getAttributes method
of *DocPrintJob*, 784
of *Node*, 170, 174
of *PrintService*, 784
getAttribute*Xxx* methods (*XMLStreamReader*), 209, 212
getAuthority method (URI), 261
getAutoCommit method (*Connection*), 355–356
getAutoCreateRowSorter method (JTable), 596
getAvailableCurrencies method (Currency), 403
getAvailableLocales method
of Collator, 412
of Locale, 31
of NumberFormat, 391, 396, 398
getAvailableZoneIds method (ZoneId), 375
getAverage method (*Xxx*SummaryStatistics), 27, 30, 49
getBinaryStream method (*Blob*), 326–327
getBlob method (*ResultSet*), 326–327
getBlockSize method (Cipher), 584
GetBooleanArrayElements function (C), 816–819
GetBooleanArrayRegion function (C), 818–819
GetBooleanField function (C), 806
getBundle method (ResourceBundle), 422–425
getByName method (InetAddress), 242–243
GetByteArrayElements function (C), 816–819, 834
GetByteArrayRegion function (C), 818–819
GetByteField function (C), 806
getBytes method (*Blob*), 326–327
getCandidateLocales method (ResourceBundle.Control), 423
getCategory method (*Attribute*), 778, 783
getCellEditorValue method (*CellEditor*), 620, 622–623, 630
getCellRenderer method (JTable), 619
getCellSelectionEnabled method (JTable), 614
getChannel method
of File*Xxx*Stream, 136
of RandomAccessFile, 136
getChar method (ByteBuffer), 133, 138
getCharacterStream method (*Clob*), 326–327

GetCharArrayElements function (C), 816–819
GetCharArrayRegion function (C), 818–819
getCharContent method (SimpleJavaFileObject), 467
GetCharField function (C), 806
getChild method (*TreeModel*), 663–665, 670
getChildAt method (*TreeNode*), 648
getChildCount method
  of *TreeModel*, 663–665, 670
  of *TreeNode*, 649
getChildNodes method (*Node*), 167, 174
getClassLoader method (Class), 535, 544
getClip method (Graphics), 711, 752
getClob method (*ResultSet*), 326–327
getCollationKey method (Collator), 410, 413
getColorModel method (BufferedImage), 735, 738
getColumn method (*TableColumnModel*), 601, 614
getColumnClass method (*TableModel*), 600, 613
getColumnCount method
  of *ResultSetMetaData*, 344
  of *TableModel*, 596–597, 600
getColumnModel method (JTable), 601, 613
getColumnName method (AbstractTableModel), 597–598, 600
getColumnNumber method
  of *Diagnostic*, 466
  of SAXParseException, 184
getColumnSelectionAllowed method (JTable), 614
getColumnType method (*TableModel*), 620
getColumn*Xxx* methods (*ResultSetMetaData*), 344, 353
getCommand method (*RowSet*), 341
getCompactNumberInstance method (NumberFormat), 398
getConcurrency method (*ResultSet*), 332, 334, 336
getConnection method (DriverManager), 305, 307, 316, 360
getConnectTimeout method (URLConnection), 268
getContent method (URLConnection), 269
getContentEncoding, getContentType methods (URLConnection), 262, 265, 269, 274
getContentLength method (URLConnection), 262, 265, 268
getContext method (ScriptEngine), 448
getContextClassLoader method (Thread), 537, 545
getCount method (*Xxx*SummaryStatistics), 30, 49
getCountry method (Locale), 35, 394
getCrc method (ZipEntry), 88
getCurrencyCode method (Currency), 403
getCurrencyInstance method (NumberFormat), 395, 398, 402
getData method (*CharacterData*), 169, 175
getDataElements method
  of ColorModel, 739
  of Raster, 735, 738
getDataSize method (DataTruncation), 314
getDate method
  of *ResultSet*, 309, 311
  of URLConnection, 262, 265, 269
getDayOf*Xxx* methods
  of LocalDate, 368, 370
  of ZonedDateTime, 378
getDays method (Period), 371
getDeclaredAnnotations method (*AnnotatedElement*), 475
getDecomposition method (Collator), 413
getDefault method (Locale), 392–394
getDefaultEditor method (JTable), 628
getDefaultFractionDigits method (Currency), 403
getDefaultName method (NameCallback), 561
getDefaultRenderer method (JTable), 619, 628
getDescent method (TextLayout), 712
getDiagnostics method (DiagnosticCollector), 466
GetDirectBuffer*Xxx* functions (C), 818
getDisplayCountry, getDisplayLanguage methods (Locale), 32, 394
getDisplayName method
  of DayOfWeek, Month, 381, 405
  of Locale, 392–394, 396
getDocumentElement method (*Document*), 166, 174
getDoInput, getDoOutput methods (URLConnection), 267
getDouble method
  of ByteBuffer, 133, 138
  of *ResultSet*, 309, 311
GetDoubleArrayElements function (C), 816–819
GetDoubleArrayRegion function (C), 818–819
GetDoubleField function (C), 802, 806
getEnclosedElements method (*TypeElement*), 489
getEngine*Xxx* methods (ScriptEngineManager), 444–445
getEntry method (ZipFile), 88

getErrorCode method (SQLException), 312, 314
getErrorStream method (HttpURLConnection), 274, 278
getErrorWriter method (*ScriptContext*), 448
getExpiration method (URLConnection), 262, 265, 269
getExtensions method (*ScriptEngineFactory*), 445
GetFieldID function (C), 802, 806
getFields method (Class), 665
getFileForOutput method (ForwardingJavaFileManager), 467
getFileName method (StackTraceElement), 116–117
getFilePointer method (RandomAccessFile), 81, 85
getFileSuffixes method (ImageReaderWriterSpi), 731
getFillsViewportHeight method (JTable), 596
getFirstChild method (*Node*), 169, 174
getFloat method (ByteBuffer), 133, 138
GetFloatArrayElements function (C), 816–819
GetFloatArrayRegion function (C), 818–819
GetFloatField function (C), 806
getFontRenderContext method (Graphics2D), 710, 712
getFormatNames method (ImageReaderWriterSpi), 731
getFragment method (URI), 261
getHeader*Xxx* methods (URLConnection), 262–265, 268
getHeight method
  of ImageReader, 725, 731
  of PageFormat, 752, 759
getHost method (URI), 261
getHost*Xxx* methods (InetAddress), 243
getHour method
  of LocalTime, 374
  of ZonedDateTime, 379
getIdentifier method (Entry), 616
getIfModifiedSince method (URLConnection), 268
getImageable*Xxx* methods (PageFormat), 752, 759
getImage*Xxx*By*Xxx* methods (ImageIO), 723, 730
getIndex method (DataTruncation), 314
getIndexOfChild method (*TreeModel*), 663, 670
getInputStream method
  of Socket, 239–240, 244
  of URLConnection, 262, 269, 272, 274
  of ZipFile, 88
getInstance method
  of AlphaComposite, 715, 721
  of Cipher, 579, 584
  of Collator, 412
  of Currency, 402–403
  of KeyGenerator, 585
  of Locale, 408
  of MessageDigest, 564–566
getInt method
  of ByteBuffer, 133, 138
  of *ResultSet*, 309, 311
GetIntArrayElements function (C), 816–819
GetIntArrayRegion function (C), 818–819
getInterface method (*Invocable*), 450
GetIntField function (C), 802, 806, 834
getISOCountries method (Locale), 392, 394
getISOLanguages method (Locale), 392
getJavaFileObjectsFrom*Xxx* methods (*StandardJavaFileManager*), 466
getJDBC*Xxx*Version methods (*DatabaseMetaData*), 352
getKeys method (ResourceBundle), 425–426
getKind method (*Diagnostic*), 466
getLanguage method (Locale), 394
getLargeUpdateCount method (*Statement*), 310
getLastChild method (*Node*), 169–170, 174
getLastModified method (URLConnection), 262, 265, 269
getLastPathComponent method (TreePath), 642, 648
getLastSelectedPathComponent method (JTree), 642, 648
getLeading method (TextLayout), 712
getLength method
  of *Attributes*, 208
  of *NamedNodeMap*, 175
  of *NodeList*, 167, 175
getLineNumber method
  of *Diagnostic*, 466
  of SAXParseException, 184
getLocale method (MessageFormat), 415
getLocalHost method (InetAddress), 242–243
getLocalName method
  of *Attributes*, 208
  of *Node*, 202
  of *XMLStreamReader*, 212
getLong method (ByteBuffer), 133, 138
GetLongArrayElements function (C), 816–819
GetLongArrayRegion function (C), 818–819

GetLongField function (C), 806
getMax method (*Xxx*SummaryStatistics), 27, 31, 49
getMaxConnections method (*DatabaseMetaData*), 352
getMaxStatements method (*DatabaseMetaData*), 311, 352
getMessage method (*Diagnostic*), 466
getMetaData method
 of *Connection*, 343, 352
 of *ResultSet*, 344, 353
getMethodCallSyntax method (*ScriptEngineFactory*), 449
GetMethodID function (C), 811, 814
getMimeTypes method (*ScriptEngineFactory*), 445
getMIMETypes method (ImageReaderWriterSpi), 731
getMin method (*Xxx*SummaryStatistics), 31, 49
getMinute method
 of LocalTime, 374
 of ZonedDateTime, 379
getModel method (Entry), 617
getMonth, getMonthValue methods
 of LocalDate, 370
 of ZonedDateTime, 378
getMonths method (Period), 371
getMoreResults method (*Statement*), 329–330
getName method
 of *Attribute*, 783
 of Field, 665
 of NameCallback, 561
 of *Principal*, 553
 of PrintService, 771
 of *XMLStreamReader*, 212
 of ZipEntry, 88
 of ZipFile, 88
getNames method (*ScriptEngineFactory*), 445
getNamespaceURI method (*Node*), 202
getNano method
 of LocalTime, 374
 of ZonedDateTime, 379
getNextEntry method (ZipInputStream), 85–87
getNextException method (SQLException), 312, 314
getNextSibling method (*Node*), 169–170, 174
getNextWarning method (SQLWarning), 314
getNode*Xxx* methods (*Node*), 170, 174, 202
getNumberInstance method (NumberFormat), 395, 398
getNumeric*Xxx* methods (Currency), 403
getNum*Xxx* methods (ImageReader), 724, 731
getObject method
 of ResourceBundle, 424–425
 of *ResultSet*, 309, 311
GetObjectArrayElement function (C), 816, 819
GetObjectClass function (C), 802–803
GetObjectField function (C), 802, 806
getOffset method (ZonedDateTime), 379
getOrientation method (PageFormat), 759
getOriginatingProvider method
 of ImageReader, 723, 731
 of ImageWriter, 732
getOutputSize method (Cipher), 584
getOutputStream method
 of Socket, 240, 244
 of URLConnection, 262, 269, 272
getOwner method (Files), 123
getPageCount method (Banner), 760
getPageSize method (*CachedRowSet*), 342
getParameter method (DataTruncation), 314
getParent method
 of ClassLoader, 544
 of *Path*, 116–117
 of *TreeNode*, 648, 650
getParentNode method (*Node*), 174
getPassword method
 of PasswordCallback, 561
 of *RowSet*, 341
getPath method
 of FileSystem, 130–131
 of TreeSelectionEvent, 662
 of URI, 261
getPaths method (TreeSelectionEvent), 656, 662
getPathToRoot method (DefaultTreeModel), 644
getPercentInstance method (NumberFormat), 395, 398
getPixel, getPixels methods (Raster), 734, 739
getPointCount method (ShapeMaker), 681
getPort method (URI), 261
getPreviousSibling method (*Node*), 174
getPrincipals method (Subject), 553
getPrinterJob method (PrinterJob), 749, 758
getPrintService method (StreamPrintServiceFactory), 773
getPrompt method
 of NameCallback, 561
 of PasswordCallback, 561
getQName method (*Attribute*), 208
getQualifiedName method (*TypeElement*), 489

getQuery method (URI), 261
getRaster method (BufferedImage), 733, 738
getReader method (*ScriptContext*), 448
getReader*Xxx* methods (ImageIO), 723, 730
getReadTimeout method (URLConnection), 268
getRequestProperties method (URLConnection), 268
getResourceAsStream method (Class, Module), 517
getResponseCode method (HttpURLConnection), 275
getResultSet method (*Statement*), 310
getRGB method
  of Color, 740
  of ColorModel, 735, 739
getRoot method
  of *Path*, 116–117
  of *TreeModel*, 663–665, 670
getRotateInstance method (AffineTransform), 706, 708
getRow method (*ResultSet*), 333, 337
getRowCount method (*TableModel*), 596–597, 600
getRowSelectionAllowed method (JTable), 614
getRow*Xxx* methods (JTable), 613
getSavepoint*Xxx* methods (*Savepoint*), 356
getScaleInstance method (AffineTransform), 706, 708
getSecond method
  of LocalTime, 374
  of ZonedDateTime, 379
getSeconds method (Duration), 363
getSelectedColumns method (JTable), 604, 608
getSelectedRows method (JTable), 604
getSelectionModel method (JTable), 613
getSelectionPath method (JTree), 642, 648, 661
getSelectionPaths method (JTree), 656, 661
getShearInstance method (AffineTransform), 706, 708
getShort method (ByteBuffer), 133, 138
GetShortArrayElements function (C), 816–819
GetShortArrayRegion function (C), 818–819
GetShortField function (C), 806
getSimpleName method (*Element*), 489
getSize method (ZipEntry), 88
getSource method (*Diagnostic*), 466
getSQLState method (SQLException), 312, 314
getSQLStateType method (SQLException), 312
getStandardFileManager method (*JavaCompiler*), 465
GetStaticFieldID, GetStatic*Xxx*Field functions (C), 805–806
GetStaticMethodID function (C), 810, 815
getStrength method (Collator), 412
getString method
  of ResourceBundle, 423, 425
  of *ResultSet*, 309, 311
getStringArray method (ResourceBundle), 426
GetStringChars, GetStringLength functions (C), 798
GetStringRegion function (C), 797
GetStringUTFChars function (C), 796, 798–799, 834
GetStringUTFLength, GetStringUTFRegion functions (C), 797
getStringValue method (Entry), 617
getSubject method (LoginContext), 552
getSubString method (*Clob*), 326–327
getSum method (*Xxx*SummaryStatistics), 30, 49
GetSuperclass function (C), 846
getSymbol method (Currency), 403
getSystemClassLoader method (ClassLoader), 544
getSystemJavaCompiler method (ToolProvider), 457
getTableCellEditorComponent method (*TableCellEditor*), 621–623, 629
getTableCellRendererComponent method (*TableCellRenderer*), 617, 629
getTableHeader method (JTable), 594
getTableName method (*CachedRowSet*), 342
getTables method (*DatabaseMetaData*), 343, 352
getTagName method (*Element*), 167, 174, 202
getTask method (*JavaCompiler*), 457–458, 465
Getters/setters, generated automatically, 492
getText method (*XMLStreamReader*), 212
getTimeZone method (TimeZone), 385
getTransferSize method (DataTruncation), 314
getTranslateInstance method (AffineTransform), 706, 709, 711
getTreeCellRendererComponent method
  of DefaultTreeCellRenderer, 652–653
  of TreeCellRenderer, 652–654
getType method
  of Field, 665
  of *ResultSet*, 332, 336
getUpdateCount method (*Statement*), 310, 330
getURI method (*Attribute*), 208
getURL method (*RowSet*), 341

getURLs method (URLClassLoader), 537
getUserInfo method (URI), 261
getUsername method (*RowSet*), 341
getValue method
  of *Attributes*, 208
  of Copies, 779
  of Entry, 617
  of Win32RegKey, 832–833
getValueAt method (*TableModel*), 596–597, 600, 619
getValueCount method (Entry), 617
getVendorName, getVersion methods (IIOServiceProvider), 723, 731
getWarnings method (Connection, ResultSet, Statement), 314
getWidth method
  of ImageReader, 725, 731
  of PageFormat, 752, 759
getWriter method (*ScriptContext*), 448
getWriter*Xxx* methods (ImageIO), 723, 730
getYear method
  of LocalDate, 370
  of ZonedDateTime, 378
getYears method (Period), 371
GIF format, 722
  animated, 724
  image manipulations on, 744
  printing, 769
GlassFish server, 359
Glob patterns, 126–127
GlobalSign, 574
GMail, 287–288
Gnu C compiler, 789–790
Gödel's theorem, 546
Google Maps, 271
GradientPaint class, 701–702
  constructor, 702–703
  cyclic parameter, 702
Graphic Java™ (Geary), 592, 630
Graphics class, 671–672
  draw*Xxx*, fill*Xxx* methods, 674
  get/setClip methods, 710–711, 752
Graphics2D class, 672
  clip method, 672, 710–712, 752
  draw method, 673–675, 692–693
  fill method, 673–674, 692
  getFontRenderContext method, 710, 712
  rotate method, 705, 709
  scale method, 704–705, 709
  setComposite method, 673, 715, 721
  setPaint method, 672, 701–703
  setRenderingHint, setRenderingHints methods, 672
  setStroke method, 672, 693, 701
  setTransform method, 707, 709
  shear method, 705, 709
  transform method, 673, 707, 709
  translate method, 705, 709, 761
Greenwich Royal Observatory, 362, 375
Gregorian calendar reform, 371
GregorianCalendar class
  toZonedDateTime method, 384–385
Groovy programming language, 444, 451–452
group method
  of Matcher, 157
  of *MatchResult*, 158
groupCount method (Matcher), 157
Grouping, 35–36
  classifier functions of, 35
  reducing to numbers, 36
groupingBy method (Collectors), 35–40
groupingByConcurrent method (Collectors), 35–36, 50
&gt;, entity reference, 164
GUI (Graphical User Interface), scripting events for, 451–456

H

\h, \H, in regular expressions, 146
Half-closing connections, 251–252
Handbook of Applied Cryptography, The (Menezes et al.), 567
handle method (*CallbackHandler*), 561
handleGetObject method (ResourceBundle), 425–426
Handles (Swing), 635, 652
hash/Digest.java, 565
hashCode method (*Annotation*), 476
HashSet class
  readObject, writeObject methods, 102
Hash*Xxx*AttributeSet classes, 750, 777
Haskell programming language, 444
hasMoreElements method (*Enumeration*), 832–835
hasNext method (*XMLStreamReader*), 211
hasRemaining method (Buffer), 137
Header information, from server, 262
header method (HttpRequest.Builder), 286

Headers (Swing tables)
  rendering, 619
  scrolling, 593
headers method (HttpResponse), 281, 286
helloNative/HelloNative.c, 789
helloNative/HelloNative.h, 788
helloNative/HelloNative.java, 787
helloNative/HelloNativeTest.java, 790
Hex editors
  creating class files in, 546
  modifying bytecodes with, 548
Hidden commands, in XML comments, 165
Hosts, 242–243
HTML (HyperText Markup Language)
  attributes in, 162, 164
  end and empty tags in, 162
  forms in, 269
  generating from XML files, 223–226
  mixing with JSP, 461
  printing, 769
  vs. XML, 161–162
HTTP (Hypertext Transfer Protocol), 295
  redirects between HTTPS and, 275
  request headers in, 263–264
http: (URI scheme), 260
HttpClient class, 279–286
  enabling logging for, 282
  newBuilder method, 279, 281, 285
  newHttpClient method, 279, 285
  send method, 285
  sendAsync method, 281, 285
HttpClient.Builder class
  build method, 279–280, 285
  executor method, 285
  followRedirects method, 279, 285
HttpHeaders class
  firstValue method, 281, 286
  map method, 281, 286
HttpRequest class
  newBuilder method, 279, 281, 286
HttpRequest.Builder class
  build method, 286
  DELETE method, 286
  GET method, 279, 286
  header method, 286
  POST method, 280, 286
  PUT method, 286
  uri method, 279–281, 286
HttpResponse class
  methods of, 280, 286
HttpResponse.BodyHandlers class
  discarding method, 281
  ofString method, 280–281
HTTPS (Hyper Text Transfer Protocol Secure), 275
https: (URI scheme), 260
HttpURLConnection class, 274
  getErrorStream method, 274, 278
  getResponseCode method, 275
  setInstanceFollowRedirects method, 275

I

I (int), type code, 96, 806
I/O streams. *See* Input streams, Output streams
IANA (Internet Assigned Numbers Authority), 375
IBM, 159
  DB2 database, 301
  DOM parser, 166
IBM437 encoding, 419
ICC profiles, 734
Icons
  in column headers, 619
  in table cells, 617
  in trees, 638–639, 651
ID, IDREF, IDREFS attribute types (DTDs), 180–181
Identity (do-nothing) transformation, 216
identity method (*Function*), 31
Identity values, 42
IDs, uniqueness of, 181, 189
IETF BCP 47, 390, 393
ifPresent method
  of Optional, 18–19
  of Optional*Xxx*, 48
ifPresentOrElse method (Optional), 19
  of Optional, 19
IIOImage class, 725, 732
IIOServiceProvider class
  get*Xxx* methods, 723, 731
IllegalAccessException, 665
IllegalArgumentException, 211, 821
IllegalStateException, 31, 724
ImageInputStream class, 724
  "Seek forward only" mode, 724

ImageIO class
  createImage*Xxx*Stream methods, 724, 730
  determining image type, 722–723
  getImage*Xxx*By*Xxx* methods, 723, 730
  getReader*Xxx*, getWriter*Xxx* methods, 723, 730
  read, write methods, 722, 729
imageIO/ImageIOFrame.java, 726
ImageOutputStream interface, 725
imageProcessing/ImageProcessingFrame.java, 744
ImageReader class, 722–723
  getHeight method, 725, 731
  getNum*Xxx* methods, 724, 731
  getOriginatingProvider method, 723, 731
  getWidth method, 725, 731
  read, readThumbnail methods, 731
  setInput method, 730
ImageReaderWriterSpi class
  get*Xxx* methods, 731
Images
  blurring, 742
  color values of, 735
  edge detection of, 743
  filtering, 740–749
  getting size of, before reading, 725
  incremental rendering of, 732
  manipulating, 732–749
  metadata in, 725
  multiple, in a file, 724–732
  printing, 749–759, 769, 773
  raster, 721–749
    constructing from pixels, 733–740
    readers/writers for, 722–732
  rotating, 741
  superimposing, 712–713
  thumbnails for, 725
  vector, 671–721
ImageWriter class, 722, 725
  canInsertImage method, 725, 732
  getOriginatingProvider method, 732
  setOutput method, 732
  write, writeInsert methods, 725, 732
implements specification, 480
#IMPLIED attribute (DTD), 181
import statement, 538
INCLUDE environment variable, 830
include method (RowFilter), 607, 616
Incremental rendering, 732
Indexed color model, 742
IndexOutOfBoundsException, 724, 731
InetAddress class
  get*Xxx* methods, 242–243
inetAddress/InetAddressTest.java, 242
InetSocketAddress class
  isUnresolved method, 259
Inheritance trees, 312
@Inherited annotation, 483, 487
init method
  of Cipher, 584
  of KeyGenerator, 585
Initialization blocks, for shared libraries, 792
initialize method (*LoginModule*), 562
INPUT_STREAM class (DocFlavor), 770
Input streams, 56–78
  as input source, 166
  buffered, 65–67
  byte processing in, 65
  byte-oriented, 56
  chaining, 65
  closing, 57
  encoding for, 68
  filters for, 64–67
  hierarchy of, 59–63
  keeping open, 251
  objects in, 89–114
  redirecting, 447–448
  Unicode, 56
InputSource class, 184
InputStream class, 56–59, 61
  available method, 57–58
  close method, 57–58
  mark method, 58
  markSupported method, 59
  nullInputStream method, 59
  read method, 56–58
  readAllBytes, readNBytes methods, 56, 58
  reset method, 59
  skip, skipNBytes methods, 58
  transferTo method, 58
InputStreamReader class, 68
INSERT statement (SQL), 300
  autogenerated keys and, 331
  executing, 308, 310, 326
  in batch updates, 355
  vs. methods of *ResultSet*, 335
insertNodeInto method (DefaultTreeModel), 643, 649

insertRow method (*ResultSet*), 335, 337
Instance fields
  accessing from native code, 801–805
  annotating, 479
instanceof keyword, 480
Instant class, 362
  compareTo method, 363
  equals method, 363
  from method, 384–385
  immutability of, 363
  legacy classes and, 384–385
  minus, minus*Xxx* methods, 365–366
  now method, 363, 365
  plus, plus*Xxx* methods, 365–366
Instrumentation API, 499
int type
  printing, 69
  storing, 79
  streams of, 43–49
  type code for, 96, 806
  vs. C types, 793
  writing in binary format, 78
IntBuffer class, 139
INTEGER data type (SQL), 301, 357
IntegerSyntax class, 779
IntelliJ IDE, 102
@interface declaration, 469, 475
Interfaces
  accessing script classes with, 450
  annotating, 479, 483
  implementing in script engines, 449
Internationalization, 387–442
Internet Engineering Task Force, 390
Interpolation, 741
  for gradients, 701–702
  strategies of, 741
  when transforming images, 741
Interruptible sockets, 252–259
interruptible/InterruptibleSocketTest.java, 254
intersect method (Area), 692
ints method
  of Random, 45
  of RandomGenerator, 48
  of SplittableRandom, 52
IntStream interface, 43–49
  average method, 44, 46
  boxed method, 44, 46
  mapToInt method, 44
  max, min methods, 44, 46
  of method, 44, 46
  range, rangeClosed methods, 44, 46
  sum, summaryStatistics methods, 44, 46
  toArray method, 44, 46
IntSummaryStatistics class, 27, 30–31, 44, 49
intValue method (Number), 396
Invalid pointers (C, C++), 786
InvalidObjectException, 110
InvalidPathException, 115
Invocable interface, 448
  getInterface method, 450
  invoke*Xxx* methods, 448, 450
Invocation API, 825–830
invocation/InvocationTest.c, 826
IOException, 62, 239
IP addresses, 237, 242–243
IPP (Internet Printing Protocol) 1.1, 783
IPv6 addresses, 242
isAfter method
  of LocalDate, 371
  of LocalTime, 374
  of ZonedDateTime, 379
isAfterLast method (*ResultSet*), 333, 337
isAnnotationPresent method (*AnnotatedElement*), 474
IsAssignableFrom function (C), 834, 846
isBefore method
  of LocalDate, 371
  of LocalTime, 374
  of ZonedDateTime, 379
isBeforeFirst method (*ResultSet*), 333, 337
isCellEditable method
  of AbstractCellEditor, 622
  of AbstractTableModel, 619
  of *CellEditor*, 630
  of DefaultTableModel, 619
  of *TableModel*, 600, 619
isCharacters method (*XMLStreamReader*), 212
isClosed method
  of *ResultSet*, 311
  of Socket, 241
  of *Statement*, 310
isConnected method (Socket), 241
isDirectory method
  of *BasicFileAttributes*, 124
  of ZipEntry, 88
isEchoOn method (PasswordCallback), 561
isEndElement method (*XMLStreamReader*), 212
isExecutable method (Files), 123–124

isFirst method (*ResultSet*), 333, 337
isGroupingUsed method (NumberFormat), 399
isHidden method (Files), 123–124
isIgnoringElementContentWhitespace method
 (DocumentBuilderFactory), 185
isInputShutdown method (Socket), 252
isLast method (*ResultSet*), 333, 337
isLeaf method
 of DefaultTreeModel, 639
 of *TreeModel*, 640, 663, 671
 of *TreeNode*, 639–640
isLeapYear method (LocalDate), 371
isNamespaceAware method
 of DocumentBuilderFactory, 202
 of SAXParserFactory, 207
isNegative method (Duration), 366
ISO 216 standard, 425
ISO 639-1 standard, 390, 394
ISO 3166-1 standard, 390, 394
ISO 4217 standard, 401, 403
ISO 8601 standard, 328, 484
ISO 8859-1 standard, 68, 77
isOutputShutdown method (Socket), 252
isParseIntegerOnly method (NumberFormat),
 399
isPresent method (Optional), 20–23
isReadable method (Files), 123–124
isRegularFile method
 of *BasicFileAttributes*, 124
 of Files, 123–124
isShared method (FileLock), 142
isStartElement method (*XMLStreamReader*), 212
isSymbolicLink method
 of *BasicFileAttributes*, 124
 of Files, 123–124
isUnresolved method (InetSocketAddress), 259
isValidating method
 of DocumentBuilderFactory, 185
 of SAXParserFactory, 207
isWhiteSpace method (*XMLStreamReader*), 212
isWritable method (Files), 123–124
isZero method (Duration), 366
item method
 of *NamedNodeMap*, 175
 of *NodeList*, 167, 175, 183
Iterable interface, 126, 457
 spliterator method, 10
iterate method (*Stream*), 5, 9, 15, 44
Iterator interface, 308
iterator method
 of *BaseStream*, 29
 of SQLException, 312, 314
 of *Stream*, 26
Iterators, 26
 converting to streams, 6, 50
 splittable, 9–10

J

J (long), type code, 96, 806
JAAS (Java Authentication and
 Authorization Service), 550–553
 configuration files in, 550, 552
 login modules in, 553–562
jaas/jaas.config, 560
jaas/JAASTest.java, 560
jaas/SimpleCallbackHandler.java, 559
jaas/SimpleLoginModule.java, 557
jaas/SimplePrincipal.java, 555
JAR files
 adding to class path, 537
 analyzing dependencies of, 528–529
 automatic registration in, 304
 file resources in, 517
 for plugins, 536
 manifest of, 86, 519
 META-INF/services directory, 526
 modular, 514–515, 520
 resources in, 421
 signing, 571
jar program, 514
jar: (URI scheme), 260
jarray type (C), 833
jarsigner program, 571
Jar*Xxx*Stream classes, 86
Java 2D API, 671–721
 affine transformations in, 706
 colors in, 734
 constructive area geometry operations
 in, 691
 features supported in, 672
 filters in, 740–749
 paint in, 701–703
 printing in, 760
 rendering pipeline in, 672–674
 sample values in, 733
 shape classes in, 674–675, 679
 strokes in, 693
 transparency in, 712–715

Java Bug Database, 742
Java EE (Java Platform, Enterprise Edition), 295
Java Platform Module System, 504–531
  migration to, 519–523
Java Plug-in, loading signed code, 577
java program
  --add-exports option, 521–523
  --add-modules option, 279
  --add-opens option, 521–523
  --illegal-access option, 521–523
  -javaagent option, 499
  jdbc.drivers property in, 304
  --module, --module-path options, 507
  -noverify option, 548
  specifying locales in, 392
Java programming language
  internationalization support in, 387
  platform-independent, 79
  vs. SQL, 321
Java Virtual Machine Specification, The, 493, 547
java.awt package, 505
java.awt.AlphaComposite API, 721
java.awt.BasicStroke API, 701
java.awt.Color API, 740
java.awt.font.TextLayout API, 712
java.awt.geom package, 102
java.awt.geom.AffineTransform API, 707–709
java.awt.geom.Arc2D.Double API, 690
java.awt.geom.Area API, 692
java.awt.geom.CubicCurve2D.Double API, 690
java.awt.geom.GeneralPath API, 690
java.awt.geom.Path2D API, 691
java.awt.geom.Path2D.Float API, 691
java.awt.geom.QuadCurve2D.Double API, 690
java.awt.geom.RoundRectangle2D.Double API, 690
java.awt.GradientPaint API, 703
java.awt.Graphics API, 711
java.awt.Graphics2D API, 674, 701, 703, 709, 712, 721
java.awt.image.AffineTransformOp API, 747
java.awt.image.BufferedImage API, 738
java.awt.image.BufferedImageOp API, 747
java.awt.image.ByteLookupTable API, 748
java.awt.image.ColorModel API, 739
java.awt.image.ConvolveOp API, 748
java.awt.image.Kernel API, 749
java.awt.image.LookupOp API, 748
java.awt.image.Raster API, 738–739
java.awt.image.RescaleOp API, 747
java.awt.image.ShortLookupTable API, 748
java.awt.image.WritableRaster API, 739
java.awt.print.PageFormat API, 759
java.awt.print.Printable API, 758
java.awt.print.PrinterJob API, 758
java.awt.TexturePaint API, 703
java.base module, 535
java.datatransfer module, 535
java.desktop module, 523–524, 535
java.instrument module, 535
java.io package, 78
java.io.Buffered*Xxx*Stream APIs, 67
java.io.Closeable API, 63
java.io.DataInput API, 80
java.io.DataOutput API, 80
java.io.File API, 118
java.io.FileInputStream API, 66, 136
java.io.FileOutputStream API, 67, 136
java.io.Flushable API, 63
java.io.InputStream API, 58–59
java.io.ObjectInputStream API, 94
java.io.ObjectOutputStream API, 94
java.io.OutputStream API, 59
java.io.PrintWriter API, 70
java.io.PushbackInputStream API, 67
java.io.RandomAccessFile API, 85, 136
java.lang, java.lang.annotation packages, 482
java.lang.annotation.Annotation API, 476
java.lang.Appendable API, 63
java.lang.CharSequence API, 48, 63
java.lang.Class API, 544
java.lang.ClassLoader API, 544–545
java.lang.Iterable API, 10
java.lang.Readable API, 63
java.lang.reflect.AnnotatedElement API, 474–475
java.lang.System API, 792
java.lang.Thread API, 545
java.logging module, 524, 535
java.management module, 535
java.management.rmi module, 535
java.naming module, 535
java.net package
  socket connections in, 240
  supporting IPv6 addresses in, 242
  URLs vs. URIs in, 260
java.net.http package, 279
java.net.http.HttpClient API, 285

java.net.http.HttpClient.Builder API, 285
java.net.http.HttpHeaders API, 286
java.net.http.HttpRequest API, 286
java.net.http.HttpRequest.Builder API, 286
java.net.http.HttpResponse API, 286
java.net.HttpURLConnection API, 278
java.net.InetAddress API, 243
java.net.InetSocketAddress API, 259
java.net.ServerSocket API, 247
java.net.Socket API, 240–241, 252
java.net.URL API, 267
java.net.URLClassLoader API, 545
java.net.URLConnection API, 267–269
java.net.URLDecoder API, 278
java.net.URLEncoder API, 278
java.nio package, 249, 252
  direct buffers in, 818
  memory mapping in, 132
java.nio.Buffer API, 137, 140–141
java.nio.ByteBuffer API, 137–138
java.nio.channels.Channels API, 259
java.nio.channels.FileChannel API, 137, 143
java.nio.channels.FileLock API, 143
java.nio.channels.SocketChannel API, 259
java.nio.CharBuffer API, 138–139
java.nio.file.attribute.BasicFileAttributes API, 124
java.nio.file.Files API, 10, 119–120, 122, 124, 129
java.nio.file.FileSystem API, 131
java.nio.file.FileSystems API, 131
java.nio.file.Path API, 117
java.nio.file.Paths API, 117
java.nio.file.SimpleFileVisitor API, 129–130
java.prefs module, 535
java.rmi module, 535
java.se module, 524
java.security package, 563
java.security.MessageDigest API, 566
java.security.Principal API, 553
java.security.sasl API, 535
java.sql package, 384
java.sql.Blob API, 327
java.sql.Clob API, 327
java.sql.Connection API, 309, 314, 325, 328, 336, 352, 356
java.sql.DatabaseMetaData API, 338, 352, 357
java.sql.DataTruncation API, 314
java.sql.DriverManager API, 307
java.sql.PreparedStatement API, 325–326
java.sql.ResultSet API, 310–311, 314, 327, 336–338, 353
java.sql.ResultSetMetaData API, 353
java.sql.Savepoint API, 356
java.sql.SQLException API, 314
java.sql.SQLWarning API, 314
java.sql.Statement API, 310, 314, 330–331, 356
java.text.CollationKey API, 413
java.text.Collator API, 412–413
java.text.Format API, 415
java.text.MessageFormat API, 415
java.text.Normalizer API, 413
java.text.NumberFormat API, 398–399
java.time package, 403–404
java.time.Duration API, 365–366
java.time.format.DateTimeFormatter API, 383–384, 407
java.time.Instant API, 365
java.time.LocalDate API, 370–371, 373, 384, 407
java.time.LocalDateTime API, 407
java.time.LocalTime API, 374, 407
java.time.Period API, 371
java.time.temporal.TemporalAdjusters API, 373
java.time.ZonedDateTime API, 378–379, 384, 407
java.util.Arrays API, 9
java.util.Collection API, 4, 53
java.util.Currency API, 403
java.util.DoubleSummaryStatistics API, 30–31, 49
java.util.function.Supplier API, 11
java.util.IntSummaryStatistics API, 30–31, 49
java.util.Locale API, 393–394
java.util.LongSummaryStatistics API, 30–31, 49
java.util.Optional API, 18–23, 26
java.util.Optional*Xxx* APIs, 48
java.util.random.RandomGenerator API, 48
java.util.regex.Matcher API, 157–158
java.util.regex.MatchResult API, 158
java.util.regex.Pattern API, 10, 156
java.util.ResourceBundle API, 425–426
java.util.Scanner API, 10, 158
java.util.Spliterators API, 9
java.util.Stream API, 43
java.util.stream.BaseStream API, 29, 53

java.util.stream.Collectors API, 30, 34, 36, 40–41
java.util.stream.DoubleStream API, 47
java.util.stream.IntStream API, 46
java.util.stream.LongStream API, 47
java.util.stream.Stream API, 4, 13–14, 16–17, 29
java.util.stream.StreamSupport API, 10
java.util.zip.ZipEntry API, 87–88
java.util.zip.ZipFile API, 88
java.util.zip.ZipInputStream API, 87
java.util.zip.ZipOutputStream API, 87
java.xml module, 535
javac program
  byte order mark in, 420
  -encoding option, 421
  passing parameters to, 457
  -XprintRounds option, 492
JavaCompiler interface
  getStandardFileManager method, 465
  getTask method, 457–458, 465
Javadoc, 485
JavaFileObject interface, 457
JavaMail, 287
javap program, 808
JavaScript programming language, 444, 451–452
javax.annotation package, 482
javax.crypto.Cipher API, 584–585
javax.crypto.Cipher*Xxx*Stream APIs, 586
javax.crypto.KeyGenerator API, 585
javax.crypto.spec.SecretKeySpec API, 585
javax.imageio package, 722
javax.imageio.IIOImage API, 732
javax.imageio.ImageIO API, 729–730
javax.imageio.ImageReader API, 730–731
javax.imageio.ImageWriter API, 732
javax.imageio.spi.IIOServiceProvider API, 731
javax.imageio.spi.ImageReaderWriterSpi API, 731
javax.print.attribute.Attribute API, 783
javax.print.attribute.AttributeSet API, 783–784
javax.print.DocPrintJob API, 772, 784
javax.print.PrintService API, 772, 784
javax.print.PrintServiceLookup API, 771
javax.print.SimpleDoc API, 772
javax.print.StreamPrintServiceFactory API, 773
javax.script.Bindings API, 447
javax.script.Compilable API, 451
javax.script.CompiledScript API, 451
javax.script.Invocable API, 450
javax.script.ScriptContext API, 448
javax.script.ScriptEngine API, 447–448
javax.script.ScriptEngineFactory API, 445
javax.script.ScriptEngineManager API, 445, 447
javax.security.auth.callback.CallbackHandler API, 561
javax.security.auth.callback.*Xxx*Callback APIs, 561
javax.security.auth.login.LoginContext API, 552
javax.security.auth.spi.LoginModule API, 562
javax.security.auth.Subject API, 553
javax.sql package, 359
javax.sql.rowset package, 339
javax.sql.RowSet API, 341
javax.sql.rowset.CachedRowSet API, 342
javax.sql.rowset.RowSetFactory API, 342
javax.sql.rowset.RowSetProvider API, 342
javax.swing.CellEditor API, 630
javax.swing.DefaultCellEditor API, 629
javax.swing.DefaultRowSorter API, 615
javax.swing.event.TreeModelEvent API, 671
javax.swing.event.TreeModelListener API, 671
javax.swing.event.TreeSelectionEvent API, 662
javax.swing.event.TreeSelectionListener API, 662
javax.swing.JComponent API, 641
javax.swing.JTable API, 596, 613–614, 628
javax.swing.JTree API, 640, 648, 661
javax.swing.ListSelectionModel API, 615
javax.swing.RowFilter API, 616
javax.swing.RowFilter.Entry API, 616–617
javax.swing.table.TableCellEditor API, 629
javax.swing.table.TableCellRenderer API, 629
javax.swing.table.TableColumn API, 615, 629
javax.swing.table.TableColumnModel API, 614
javax.swing.table.TableModel API, 600, 613
javax.swing.table.TableRowSorter API, 615
javax.swing.table.TableStringConverter API, 616
javax.swing.tree.DefaultMutableTreeNode API, 641, 654
javax.swing.tree.DefaultTreeCellRenderer API, 654
javax.swing.tree.DefaultTreeModel API, 641, 649
javax.swing.tree.MutableTreeNode API, 640
javax.swing.tree.TreeCellRenderer API, 654
javax.swing.tree.TreeModel API, 640, 670–671
javax.swing.tree.TreeNode API, 640, 648–649
javax.swing.tree.TreePath API, 648
javax.tools.Diagnostic API, 466
javax.tools.DiagnosticCollector API, 466

javax.tools.ForwardingJavaFileManager API, 467
javax.tools.JavaCompiler API, 465
javax.tools.JavaCompiler.CompilationTask API, 466
javax.tools.SimpleJavaFileObject API, 467
javax.tools.StandardJavaFileManager API, 466
javax.tools.Tool API, 465
javax.xml.catalog.CatalogFeatures API, 184
javax.xml.catalog.CatalogManager API, 184
javax.xml.catalog.files system property, 178
javax.xml.parsers.DocumentBuilder API, 173, 183, 215
javax.xml.parsers.DocumentBuilderFactory API, 173, 185, 202
javax.xml.parsers.SAXParser API, 207
javax.xml.parsers.SAXParserFactory API, 207
javax.xml.stream.XMLInputFactory API, 211
javax.xml.stream.XMLOutputFactory API, 221
javax.xml.stream.XMLStreamReader API, 211–212
javax.xml.stream.XMLStreamWriter API, 221–222
javax.xml.transform.dom.DOMResult API, 233
javax.xml.transform.dom.DOMSource API, 216
javax.xml.transform.sax.SAXSource API, 233
javax.xml.transform.stream.StreamResult API, 216
javax.xml.transform.stream.StreamSource API, 233
javax.xml.transform.Transformer API, 216
javax.xml.transform.TransformerFactory API, 216, 232
javax.xml.xpath.XPath API, 199
javax.xml.xpath.XPathEvaluationResult API, 199
javax.xml.xpath.XPathFactory API, 199
JAXB (Java Architecture for XML Binding), 516
JAXP (Java API for XML Processing) library, 166, 537
jboolean type (C), 793
jbooleanArray type (C), 816
JButton class, 452
jbyte type (C), 793
jbyteArray type (C), 816
jchar type (C), 793, 795
jcharArray type (C), 816
JCheckBox class, 619
jclass type (C), 811
JComboBox class, 619
JCommander framework, 469
JComponent class
  paint method, 617, 672
  paintComponent method, 672
  putClientProperty method, 636, 641
JDBC API, 291–360
  configuration of, 301–307
  debugging, 306
  design of, 292–295
  specification for, 293
  tracing, 306
  uses of, 294–295
  versions of, 291
JDBC API Tutorial and Reference (Fisher et al.), 336, 358
JDBC drivers
  escape syntax in, 328–329
  JAR files for, 302
  registering classes for, 304
  scrollable/updatable result sets in, 332
  types of, 293–294
JDBC/ODBC bridge, 293
JdbcRowSet interface, 339
jdeprscan program, 484
jdeps program, 528–529
JDK (Java Development Kit)
  DOM parser, 166
  keytool program, 569
  obsolete features in, 504
  serialver program, 107
  src.jar file, 826
  SunJCE ciphers, 579
jdk.incubator.http package, 279
jdouble type (C), 793
jdoubleArray type (C), 816
JEP 290, JEP 415, 114
jfloat type (C), 793
jfloatArray type (C), 816
jimage command, 530
jint type (C), 793
jintArray type (C), 816
JLabel class, 651–652
jlink program, 530
jlong type (C), 793
jlongArray type (C), 816
JMOD files, 530, 535
jmod program, 530
JNDI service, 359, 537
JndiLoginModule class, 551
JNI (Java Native Interface), 786–846
  accessing:
    array elements in, 816–819
    functions in C++, 796
  calling convention in, 796

debugging mode of, 826
error handling in, 819–825
invoking Java methods in, 809–815
online documentation for, 797
JNI_CreateJavaVM function (C), 825–826, 830
JNI_OnLoad, JNI_OnUnload methods (C), 792
jni.h file, 793
JNICALL, JNIEXPORT macros, 788
JobAttributes class (obsolete), 783
jobject type (C), 811, 816, 833
jobjectArray type (C), 816
Join styles, 694
joining method (Collectors), 27, 30
JoinRowSet interface, 339
JPanel class, 452, 753
JPEG format, 722
image manipulations on, 744
printing, 769
reading, 723
js.properties file, 451
JScrollPane class, 593
JSF (JavaServer Faces), 269
JShell, loading modules into, 515
jshort type (C), 793
jshortArray type (C), 816
JSON-B (JSON Binding), 516, 518
JSP (JavaServer Pages), 461–467
jstring type (C), 796, 811, 833
JTable class, 591–630
addColumn method, 608, 614
constructor, 596
convert*Xxx*IndexToModel methods, 604, 614
default rendering actions, 600
getAutoCreateRowSorter method, 596
getCellRenderer method, 619
getCellSelectionEnabled method, 614
getColumnModel method, 601, 613
getColumnSelectionAllowed method, 614
getDefaultEditor method, 628
getDefaultRenderer method, 619, 628
getFillsViewportHeight method, 596
getRowHeight, getRowMargin methods, 613
getRowSelectionAllowed method, 614
getSelectedColumns method, 604, 608
getSelectedRows method, 604
getSelectionModel method, 613
getTableHeader method, 594
installing cell editors automatically, 619
moveColumn method, 608, 614
print method, 594, 596
removeColumn method, 608, 614
resize modes of, 603
setAutoCreateRowSorter method, 594, 596, 605
setAutoResizeMode method, 603, 613
setCellSelectionEnabled method, 604, 614
setColumnSelectionAllowed method, 604, 614
setDefaultRenderer method, 618
setFillsViewportHeight method, 596
setRowHeight, setRowMargin methods, 603, 613
setRowSelectionAllowed method, 604, 614
setRowSorter method, 605, 614
JTextField class, 619
JTree class, 630–671
addTreeSelectionListener method, 655
constructor, 632, 640
getLastSelectedPathComponent method, 642, 648
getSelectionPath method, 642, 648, 661
getSelectionPaths method, 656, 661
identifying nodes, 641
makeVisible method, 644, 648
scrollPathToVisible method, 644, 648
setEditable method, 645
setRootVisible method, 638, 640
setShowsRootHandles method, 637, 640
JUnit tool, 468
Just-in-time compiler, 825
JVM (Java virtual machine)
class files in, 534
class loaders in, 535
creating, 825
embedding into native code, 825–830
specification for, 493, 547
terminating, 826
jvm pointer (C), 825–826

K

\k, in regular expressions, 147
Kerberos protocol, 550
Kernel class, 743, 749
Kernel, of a convolution, 743
Key/value pairs. *See* Property files
Keyboard, reading from, 56, 68
KeyGenerator class, 580
generateKey method, 580, 585
getInstance method, 585
init method, 585

KeyPairGenerator class, 587
KeyStoreLoginModule class, 551
Keystores, 569–571
keytool program, 569–571
Krb5LoginModule class, 551

L

L (object), type code, 96, 806
Lambda expressions, 11
Landscape orientation, 707
Language codes, 35, 390
Language Model API, 489
Language tags, 393
last method (*ResultSet*), 333, 337
lastDayOf*Xxx*, lastInMonth methods (TemporalAdjuster), 373
last*Xxx*Time methods (*BasicFileAttributes*), 124
Layout algorithm, 760
layoutPages method (Banner), 760
Lazy operations, 3, 7, 15, 154
LCD displays, 734
LD_LIBRARY_PATH environment variable, 792, 830
Leap seconds, 362
Leap years, 367, 371
Learn SQL The Hard Way (Shaw), 295
Learning SQL (Beaulieu), 295
Leaves (Swing), 630, 638, 663
  icons for, 638–639, 651
Legacy APIs, 117, 384–385
Legacy data, converting into XML, 228
length method
  of *Blob*, 327
  of *CharSequence*, 63
  of *Clob*, 327
  of RandomAccessFile, 81, 85
LIB environment variable, 830
lib/ext directory, 535
LIKE statement (SQL), 299, 329
limit method (*Stream*), 13, 50, 137
Line feed, 69, 418
  in e-mails, 287
  in regular expressions, 148
Line2D class, 674–675
Line2D.Double, Line2D.Float classes, 675, 677
lines method
  of BufferedReader, 72
  of Files, 6, 10, 51, 118
lineTo method (Path2D.Float), 680, 691
Linux operating system
  compiling invocation API, 829
  library path in, 792
  OpenSSL in, 576
  using GNU C compiler, 789
list method (Files), 124–125
ListResourceBundle class, 424
Lists, converting to streams, 50
ListSelectionModel interface
  setSelectionMode method, 604, 615
LITTLE_ENDIAN constant (ByteOrder), 138
Little-endian order, 76, 79, 133, 420
Load time, 499
loadClass method
  of ClassLoader, 537, 539
  of URLClassLoader, 536
loadLibrary method (System), 790, 792
LOBs (large objects), 326–328
  creating empty, 328
  placing in database, 326
  reading, 326
Local hosts, 242
Local names, 202
Local variables, annotating, 479
LocalDate class
  datesUntil method, 368, 371
  format method, 404
  getDayOf*Xxx* methods, 368, 370
  getMonth, getMonthValue methods, 370
  getYear method, 370
  isAfter, isBefore, isLeapYear methods, 371
  legacy classes and, 385
  minus, minus*Xxx* methods, 368, 370
  now method, 367, 370
  of method, 367, 370, 372
  parse method, 384, 404, 407
  plus, plus*Xxx* methods, 367–368, 370
  toLocalDate method, 385
  until method, 367, 370
  weekends in, 368
  with method, 372–373
  with*Xxx* methods, 370
localdates/LocalDates.java, 369
LocalDateTime class, 374
  atZone method, 375
  format method, 404
  legacy classes and, 385
  parse method, 404, 407
  toLocalDateTime method, 385

Locale class, 389–394
  constructor, 393
  debugging, 394
  forLanguageTag method, 391, 394
  getAvailableLocales method, 31
  getCountry method, 35, 394
  getDefault method, 392–394
  getDisplayCountry, getDisplayLanguage methods, 32, 394
  getDisplayName method, 392–394, 396
  getInstance method, 408
  getISOCountries method, 392, 394
  getISOLanguages method, 392
  getLanguage method, 394
  predefined objects, 391
  setDefault method, 392, 394
  toLanguageTag method, 391, 394
  toString method, 394
Locales, 388–394
  current, 415
  date and time formatting in, 403–407
  default, 380, 392
  dictionary ordering in, 408
  display names of, 32, 392–394
  formatting styles for, 381
  grouping, 35–36
  mapping names of, 31–32
  numbers in, 388–389, 395–403
  predefined, 391
  resources bundles and, 422–423
  variants in, 389, 423
LocalTime class, 373–374
  format method, 404
  get*Xxx* methods, 374
  isAfter, isBefore methods, 374
  legacy classes and, 385
  minus, minus*Xxx* methods, 373–374
  now method, 373–374
  of method, 373–374
  parse method, 404, 407
  plus, plus*Xxx* methods, 373–374
  toLocalTime method, 385
  to*Xxx*OfDay methods, 374
  with*Xxx* methods, 374
lock method (FileChannel), 141–143
Locks, 141–143
  for the tail portion of a file, 142
  shared, 142
  unlocking, 142
Log files, 420
Log messages, adding to classes, 493–499
@LogEntry annotation, 493
Logging, code generation for, 467
logging.properties file, 282
LoginContext class, 550
  constructor, 552
  getSubject method, 552
  login, logout methods, 550, 552
LoginException, 552
LoginModule interface
  documentation for, 555
  methods of, 562
Logins
  committed, 555
  modules for, 550–551
    custom, 553–562
  separating from action code, 555
Long class
  MAX_VALUE constant, 142
LONG NVARCHAR data type (SQL), 357
long type
  printing, 69
  streams of, 43–49
  type code for, 96, 806
  vs. C types, 793
  writing in binary format, 78
LONG VARCHAR data type (SQL), 357
LongBuffer class, 139
longs method
  of Random, 45
  of RandomGenerator, 48
  of SplittableRandom, 52
LongStream interface, 43–49
  average method, 44, 47
  boxed method, 44, 47
  mapToLong method, 44
  max, min methods, 44, 47
  of method, 47
  range, rangeClosed methods, 44, 47
  sum, summaryStatistics methods, 44, 47
  toArray method, 44, 47
LongSummaryStatistics class, 27, 30–31, 44, 49
Look-and-feel
  displaying trees in, 636
  handles for subtrees in, 652
  selecting multiple nodes in, 656
lookingAt method (Matcher), 157
lookup method (MethodHandles), 518

LookupOp class, 740–741
constructor, 748
lookupPrintServices method (PrintServiceLookup), 770–771
lookupStreamPrintServiceFactories method (StreamPrintServiceFactory), 773
LookupTable class, 742
LSB (least significant byte), 79
LSOutput interface, 215
LSSerializer interface, 214
writeToString method, 215
&lt;, entity reference, 164

M

Mac OS X
character encodings in, 418
OpenSSL in, 576
resources in, 421
Mac Roman encoding, 418
mail/MailTest.java, 289
main method, 534
makeShape method (ShapeMaker), 681
makeVisible method (JTree), 644, 648
Mandelbrot set, 735–736
Mangling names, 789, 806
Manifest files, 86
map method
of FileChannel, 132, 137
of HttpHeaders, 281, 286
of Optional, 19–20
of *Stream*, 11
mapping method (Collectors), 37, 41
Maps
concurrent, 32
of stream elements, 31–34, 50
mapToInt method (*Stream*), 42
mapTo*Xxx* methods (*Xxx*Stream), 44
mark method
of Buffer, 140–141
of InputStream, 58
Marker annotations, 477
markSupported method (InputStream), 59
Mastering Regular Expressions (Friedl), 149
match attribute (XSLT), 225
match/HrefMatch.java, 150
Matcher class, 149
end method, 157
find method, 157
group, groupCount methods, 157
lookingAt method, 157
matches method, 157
quoteReplacement method, 157
replaceAll method, 154, 157–158
replaceFirst method, 155, 157
results method, 158
start method, 157
matcher method (Pattern), 149, 156
matches method
of Matcher, 157
of Pattern, 149
MatchResult interface, 150
methods of, 158
Matrices, transformations of, 706
max method
of primitive streams, 44, 46–47
of *Stream*, 16–17
MAX_VALUE constant (Long), 142
maxBy method (Collectors), 37, 40
maxoccurs attribute (XML Schema), 187
MD5 algorithm, 564
Memory addresses, vs. serial numbers, 91
Memory mapping of files, 131–141
memoryMap/MemoryMapTest.java, 134
Message digests, 563–566
MessageDigest class
digest method, 565–566
extending, 564
getInstance method, 564–566
reset method, 566
update method, 564, 566
MessageFormat class, 413–418, 427
applyPattern method, 415
constructor, 415
format method, 413–415
get/setLocale methods, 415
ignoring the first limit, 417
Meta-annotations, 469, 484–488
Metadata (databases), 343–353
Metal look-and-feel
selecting multiple nodes in, 656
trees in, 635–636
Method class, 470
Method references, type use annotations in, 480
Method verification errors, 548
MethodHandle class, 847
MethodHandles class
lookup method, 518

Methods
  adding logging messages to, 493–499
  annotating, 468, 479, 483
  calling from native code, 809–815
  getters/setters, generated automatically, 492
  instance, 809–810
  mangling names of, 789, 806
  native, 785–847
  of annotation interfaces, 476
  overriding, 484
  protected, 537
  signatures of:
    generating, 808
    mangling, 806–808
  static, 810–811
Microsoft
  Active Server Pages (ASP), 269
  compiler, 789–790
    invocation API in, 829
  Notepad, 76
  ODBC API, 292
  SQL Server, 301
  Windows. *See* Windows operating system
MIME (Multipurpose Internet Mail Extensions), 722
  for print services, 770
  getting, of a file, 118
`MimeMessage` class
  methods of, 288
`min` method
  of primitive streams, 44, 46–47
  of *`Stream`*, 16–17
`minBy` method (`Collectors`), 37, 40
`minoccurs` attribute (XML Schema), 187
`minus`, `minus`*Xxx* methods
  of `Duration`, 366
  of `Instant`, 365
  of `LocalDate`, 368, 370
  of `LocalTime`, 373–374
  of `Period`, 371
  of `ZonedDateTime`, 378
`mismatch` method (`Files`), 118–119
`MissingResourceException`, 422
Miter join, miter limit, 694
Mixed content (XML), 163, 179
Modernist painting example, 222–223
Modified UTF-8, 79–80, 420
  native code and, 795–798
`Module` class
  `getResourceAsStream` method, 517
`module` keyword, 507
`module-info.class` file, 514, 519
`module-info.java` file, 507, 519
Modules, 504–531
  accessing, 515–518, 522
  automatic, 518–521
  declaration of, 507–508
  explicit, 521
  exporting packages, 510–514
  loading into JShell, 515
  migration to, 519–523
  naming, 505–506, 519
  not passing access rights, 509
  open, 517
  opening packages in, 517
  packages with the same names in, 514
  platform classes in, 535
  qualified exports of, 525
  requiring, 508–510
  service implementations and, 526
  tools for, 528–531
  unnamed, 521
  versioning, 505, 508
`Month` enumeration, 367
  `getDisplayName` method, 381, 405
`MonthDay` class, 368
`move` method (`Files`), 120–122
`moveColumn` method (`JTable`), 608, 614
`moveTo` method (`Path2D.Float`), 680–681, 691
`moveTo`*Xxx*`Row` methods (*`ResultSet`*), 335, 337
MSB (most significant byte), 79
Multiple-page printing, 758–761
`multipliedBy` method (`Duration`), 366
`MutableTreeNode` interface
  implementing, 632
  `setUserObject` method, 632, 640
MySQL database, 301

N

`\n`
  as line feed, 69, 170, 287, 418
  in regular expressions, 145–147, 156
`NameCallback` class, 554
  constructor and methods of, 561

NamedNodeMap interface
  getLength method, 175
  item method, 175
names method (Win32RegKey), 832
Namespaces, 199–202
  activating processing of, 188
  aliases (prefixes) for, 185, 201
  of attributes, 201
  of child elements, 200
  using class loaders as, 538–539
National character strings, 358
National Institute of Standards and Technology, 236, 564
native keyword, 787
Native methods, 785–847
  array elements in, 816–819
  calling from Java programs, 786–792
  class references in, 803
  compiling, 789
  enumerating keys with, 834
  error handling in, 819–825
  exceptions in, 820
  garbage collection and, 797
  instance fields in, 801–805
  invoking Java constructors in, 811
  linking to Java, 792
  naming, 787–788
  numeric parameters in, 793–795
  overloading, 788
  reasons to use, 786
  registry access functions in, 833–846
  static, 787
  static fields in, 805–806
  strings in, 795
native.encoding property, 77
native2ascii program, 423
NCHAR, NCLOB data types (SQL), 357–358
negated method (Duration), 366
Nervous text applet, 562
net.properties file, 282
Networking, 235–290
  connecting to a server, 235–243
  debugging, 235–238
  getting web data, 259–278
  HTTP client for, 279–286
  implementing servers, 244–259
  sending e-mails, 287–290
newBuffered*Xxx* methods (Files), 118–119
newBuilder method
  of HttpClient, 279, 285
  of HttpRequest, 281, 286
NewDirectByteBuffer function (C), 818
newDirectoryStream method (Files), 126, 129
newDocument method (DocumentBuilder), 213, 215, 228
newDocumentBuilder method (DocumentBuilderFactory), 166, 173, 213
newFactory method (RowSetProvider), 339, 342
newFileSystem method (FileSystems), 130–131
NewGlobalRef function (C), 803
newHttpClient method (HttpClient), 279, 285
newInputStream method
  of Channels, 259
  of Files, 118–119
newInstance method
  of DocumentBuilderFactory, 166, 173, 201
  of SAXParserFactory, 204, 207
  of TransformerFactory, 214, 216
  of XMLInputFactory, 211
  of XMLOutputFactory, 217, 221
  of XPathFactory, 195, 199
newNSInstance method (DocumentBuilderFactory), 201–202, 205
NewObject function (C), 811, 815, 820
newOutputStream method
  of Channels, 253, 259
  of Files, 118–119
newSAXParser method (SAXParserFactory), 204, 207
NewString function (C), 798
NewStringUTF function (C), 796–797, 799, 833
newTransformer method (TransformerFactory), 214, 216, 232
newXpath method (XPathFactory), 199
New*Xxx*Array functions (C), 818, 833
next method
  of *ResultSet*, 308, 310, 331
  of TemporalAdjusters, 373
  of *XMLStreamReader*, 211–212
nextElement method (*Enumeration*), 649, 832–835
nextOrSame method (TemporalAdjusters), 372–373
nextPage method (*CachedRowSet*), 340, 342
NMTOKEN, NMTOKENS attribute types (DTDs), 180–181
No-argument constructors, 101, 113

Node interface
  appendChild method, 213, 215
  getAttributes method, 170, 174
  getChildNodes method, 167, 174
  getFirstChild method, 169, 174
  getLastChild method, 169–170, 174
  getLocalName method, 202
  getNamespaceURI method, 202
  getNextSibling method, 169–170, 174
  getNode*Xxx* methods, 170, 174, 202
  getParentNode method, 174
  getPreviousSibling method, 174
  subinterfaces of, 167
nodeChanged method (DefaultTreeModel), 643, 649
NodeList interface, 167
  getLength method, 167, 175
  item method, 167, 175, 183
Nodes (Swing), 630
  adding, 643
  child, 630, 633
  collapsed, 645
  connecting lines for, 636–637
  currently selected, 642
  editing, 645, 664
  enumerating, 649–651
  expanding, 643, 645
  handles for, 635, 637, 652
  highlighting, 652
  identifying, by tree paths, 641
  making visible, 644
  parent, 630, 633
  removing, 643
  rendering, 639, 651–654
  root, 630–638
  row positions of, 643
  searching, 651, 656
  selecting, 655
  user objects for, 632, 643, 651, 656
nodesChanged method (DefaultTreeModel), 649
Nondeterministic parsing, 180
noneMatch method (*Stream*), 17
Noninterference, of stream operations, 7
@NonNull annotation, 480–481
Normalization, 409
normalize method
  of *Normalizer*, 413
  of *Path*, 116–117
Normalized color values, 733
Normalizer class, 410
  normalize method, 413
NoSuchAlgorithmException, 566, 584
NoSuchElementException, 20–21, 835
notFilter method (RowFilter), 607, 616
NotSerializableException, 102
now method
  of Instant, 363, 365
  of LocalDate, 367, 370
  of LocalTime, 373–374
  of ZonedDateTime, 378
NTLoginModule class, 551
nullInputStream method (InputStream), 59
nullOutputStream method (OutputStream), 59
NullPointerException, 821
  vs. Optional, 16
Number class
  doubleValue method, 396
  intValue method, 396
numberFilter method (RowFilter), 606, 616
NumberFormat class, 395–403
  format method, 396, 398
  get/set*Xxx*Digits methods, 399
  getAvailableLocales method, 391, 396, 398
  getCompactNumberInstance method, 398
  getCurrencyInstance method, 395, 398, 402
  getNumberInstance method, 395, 398
  getPercentInstance method, 395, 398
  is/setGroupingUsed methods, 399
  is/setParseIntegerOnly methods, 399
  parse method, 395–396, 399
  setCurrency method, 402
numberFormat/NumberFormatTest2.java, 397
Numbers
  filtering, 606–607
  floating-point, 388, 395–403
  formatting, 388–389, 395–403, 413–415
    supported locales for, 396
    with C, 793
  from grouped elements, 36
  in regular expressions, 146
  printing, 69
  random, 5, 13, 45, 581
  reading:
    from files, 65
    from ZIP archives, 66
    using locales, 395
  storing in memory, 79
  writing in binary format, 78

NUMERIC data type (SQL), 301, 357
NVARCHAR data type (SQL), 357

O

Object class
  clone method, 89, 110
Object inspection tree, 662–671
Object serialization, 89–114
  cloning with, 110–113
  file format for, 94–101
  modifying default, 101–104
  of singletons, 104–107
  serial numbers for, 90–91
ObjectInputStream class, 89
  constructor, 94
  readObject method, 89, 94, 102
ObjectInputValidation interface, 113–114
ObjectOutputStream class, 89–90
  constructor, 94
  defaultWriteObject method, 102
  writeObject method, 89, 94, 102
Object-relational mappers, 516
Objects
  cloning, 110–113
  converting to streams, 6, 24
  fingerprints of, 95
  printing, 69
  reading from an input stream, 89
  saving:
    in database, 487
    in output streams, 89–90
    in text format, 72–75
  serializable, 89–94
  transmitting over network, 91
  type code for, 96, 806
  versioning, 107–110
ODBC API, 292, 294
of method
  of *DoubleStream*, 47
  of *IntStream*, 44, 46
  of LocalDate, 367, 370, 372
  of LocalTime, 373–374
  of *LongStream*, 47
  of Optional, 21–22
  of *Path*, 115
  of Period, 371
  of *Stream*, 5, 9
  of ZonedDateTime, 375, 378
  of ZoneId, 375
ofDateAdjuster method (TemporalAdjusters), 372
ofDays method
  of Duration, 365
  of Period, 367, 371, 376
ofFile, ofFileDownload methods (BodyHandlers), 281
OffsetDateTime class, 376
ofHours method (Duration), 365
ofInstant method (ZonedDateTime), 378
ofLocalized*Xxx* methods (DateTimeFormatter), 380, 383, 403, 407
ofMillis, ofMinutes methods (Duration), 365
ofMonths method (Period), 368, 371
ofNanos method (Duration), 365
ofNullable method
  of Optional, 21–22
  of *Stream*, 6, 9, 24
ofPattern method (DateTimeFormatter), 381, 384
ofSeconds method (Duration), 365
ofString method (BodyHandlers, BodyPublishers), 280–281
ofWeeks, ofYears methods (Period), 371
oj literal (SQL), 328
open keyword, 517
open method
  of FileChannel, 132, 137
  of SocketChannel, 252, 259
openConnection method (URL), 262, 267
openOutputStream method (SimpleJavaFileObject), 467
opens keyword, 517, 525
OpenSSL toolkit, 576
openStream method (URL), 64, 259, 267
Operating system
  character encodings in, 77, 418
  default locales in, 392
  paths in, 64, 115
  resources in, 421
Operations
  associative, 42
  lazy, 3, 7, 15, 154
  stateless, 49
Optional class, 16–23
  creating values of, 21
  empty method, 21–22
  filter method, 19–20
  flatMap method, 22–26
  for empty streams, 41–42
  get method, 20–23

ifPresent method, 18–19
ifPresentOrElse method, 19
isPresent method, 20–23
map method, 19–20
of, ofNullable methods, 21–22
or method, 20
orElse method, 16, 18
orElseGet method, 18
orElseThrow method, 18, 21
stream method, 23–26
optional keyword, 551
optional/OptionalTest.java, 24
OptionalInt, OptionalDouble, OptionalLong classes, 44, 48
Oracle
JDBC, 292, 301
JVM implementation, 77, 818
ORDER BY statement (SQL), 309
order method (ByteBuffer), 133, 138
orElse, orElseGet methods (Optional*Xxx*), 48
orFilter method (RowFilter), 607, 616
org.omg.corba package, 504
org.w3c.dom package, 166
org.w3c.dom.CharacterData API, 175
org.w3c.dom.Document API, 174, 215
org.w3c.dom.Element API, 174, 215
org.w3c.dom.NamedNodeMap API, 175
org.w3c.dom.Node API, 174, 202, 215
org.w3c.dom.NodeList API, 175
org.xml.sax.Attributes API, 208
org.xml.sax.ContentHandler API, 207–208
org.xml.sax.EntityResolver API, 183
org.xml.sax.ErrorHandler API, 184
org.xml.sax.helpers.AttributesImpl API, 233
org.xml.sax.InputSource API, 184
org.xml.sax.SAXParseException API, 184
org.xml.sax.XMLReader API, 233
Orientation, 104–105
Outer joins, 328
OutOfMemoryError, 821
output element (XSLT), 225
Output streams, 56–78
buffered, 65–67
byte processing in, 65
byte-oriented, 56
closing, 57, 251
filters for, 64–67
hierarchy of, 59–63
objects in, 89–114
redirecting, 447–448
Unicode and, 56
OutputStream class, 56, 61–62, 217
close method, 59
flush method, 57, 59
nullOutputStream method, 59
write method, 57, 59
OutputStreamWriter class, 68
OverlappingFileLockException, 142
Overloading, 788
@Override annotation, 483–484

P

\p, \P, in regular expressions, 146
Package class, 470
package-info.java file, 479
Packages, 504
annotating, 479, 483
avoiding name clashes with, 538–539
exporting, 510–514
hidden, 514
opening, 517
split, 514
Packets, 240
Padding schemes, 580
Page setup dialog box, 753–754
Pageable interface
implementing, 759
objects, printing, 769
PageAttributes class (obsolete), 783
pageDialog method (PrinterJob), 753, 755, 758
PageFormat class
getHeight method, 752, 759
getImageable*Xxx* methods, 752, 759
getOrientation method, 759
getWidth method, 752, 759
Pages
measurements of, 752–753
multiple, printing, 759–769
orientation of, 707, 753, 759
Paint interface, 701–703
paint method (JComponent), 617, 672
paintComponent method
of JComponent, 672
of StrokePanel, 696
panama/PanamaDemo.java, 847
Paper
margins of, 752
sizes of, 425, 752

parallel method (*BaseStream*), 53
parallel/ParallelStreams.java, 52
parallelStream method (*Collection*), 2–4, 49–53
Parameter variables, annotating, 479
Parent nodes (Swing), 630, 633
parse method
  of DateTimeFormatter, 381
  of DocumentBuilder, 173
  of LocalDate, 384, 404, 407
  of LocalDateTime, LocalTime, 407
  of NumberFormat, 395–396, 399
  of SAXParser, 204, 207
  of *XMLReader*, 233
  of ZonedDateTime, 384, 404, 407
Parsed character data, 179
ParseException, 396, 398
Parsers, 165–175
  checking uniqueness of IDs in, 181, 189
  pull, 208
  validating in, 176
Parsing (XML), 165–175
  nondeterministic, 180
  with XML Schema, 188
partitioningBy method (Collectors), 35–36, 38
PasswordCallback class, 554
  constructor and methods of, 561
Password-protected resources, 264
Path interface, 115–118
  get*Xxx* methods, 116–117
  normalize method, 116–117
  of method, 115
  relativize method, 116–117
  resolve, resolveSibling methods, 116–117
  toAbsolutePath, toFile methods, 116–117
Path2D class
  append method, 681, 691, 711
  closePath method, 680, 691
Path2D.Double class, 675, 677
Path2D.Float class, 675, 677
  curveTo, lineTo, moveTo, quadTo methods, 680–681, 691
pathFromAncestorEnumeration method (DefaultMutableTreeNode), 650
Paths (file system), 115–118
  absolute vs. relative, 64, 115–116
  checking properties of, 123–124
  filtering, 125
  relativizing, 116
  resolving, 66, 116
  root component of, 115
  separators in, 64, 115
Paths (graphics), 680–681
Paths class, 130
  get method, 115, 117
Pattern class, 149
  asMatchPredicate, asPredicate methods, 149
  compile method, 149, 155–156
  flags, 155–158
  matcher method, 149, 156
  matches method, 149
  split method, 154, 156
  splitAsStream method, 6, 10, 154, 156
Patterns, 143–158
#PCDATA element content (DTD), 178–179
PDF format, 769
peek method (*Stream*), 15–16
PEM (Privacy Enhanced Mail), 576
Pentium processor, little-endian order in, 79
Percentages, formatting, 395–396
Performance
  of encryption algorithms, 587
  of file operations, 131–139
Period class
  get*Xxx* methods, 371
  minus, minus*Xxx* methods, 371
  of method, 371
  of*Xxx* methods, 367–368, 371, 376
  plus, plus*Xxx* methods, 371
  using for daylight savings time, 376
  with*Xxx* methods, 371
Perl programming language, 149
@Persistent annotation, 487
Personal data, 578
Picocli framework, 469
Pixels
  affine transformations on, 740
  average value of, 743
  composing, 712–721
  interpolating, 701–702, 741
  sample values of, 733
  setting individual, 733–740
Placeholders, in messages, 413–418
Plugins, loading, 536
plus, plus*Xxx* methods
  of Duration, 366
  of Instant, 365

of LocalDate, 367–368, 370
of LocalTime, 373–374
of Period, 371
of ZonedDateTime, 378
PNG format, 722
printing, 769
Point2D class, 677
Point2D.Double class, 102, 675, 677
Point2D.Float class, 675, 677
Points, in typography, 752
Policy files, 577
Polygons, 675, 681
Pools, for parallel streams, 51
populate method (*CachedRowSet*), 339, 342
Porter–Duff rules, 713–716
Portrait orientation, 753
Ports, 237
blocking, 236
in URIs, 261
position function (XPath), 227
position method (Buffer), 141
POSIX-compliant file systems, 123
PosixFileAttributes interface, 123
POST method (HttpRequest.Builder), 280, 286
POST request (HTML), 270, 272–274
building, 280
post/PostTest.java, 275
@PostConstruct annotation, 483
PostgreSQL database, 301
connecting to, 305
drivers for, 302
Postorder traversal, 650
postOrderEnumeration method (DefaultMutableTreeNode), 650, 654
PostScript format
printing, 769, 772
writing to, 772
postVisitDirectory method
of *FileVisitor*, 127
of SimpleFileVisitor, 130
Predefined character classes, 144, 146, 148
@PreDestroy annotation, 483
Predicate functions, 35
premain method (Instrumentation API), 499
preOrderEnumeration method (DefaultMutableTreeNode), 650, 654
Prepared statements, 319–326
caching, 320
executing, 320
PreparedStatement interface
clearParameters method, 325
execute*Xxx* methods, 320, 326
set*Xxx* methods, 320, 325
prepareStatement method (*Connection*), 319–320, 325, 331, 336
previous method (*ResultSet*), 332, 336
previous, previousOrSame methods (TemporalAdjusters), 373
previousPage method (*CachedRowSet*), 342
preVisitDirectory method
of *FileVisitor*, 127
of SimpleFileVisitor, 130
Primitive types
arrays of, 818
I/O in binary format in, 60
streams of, 42–49
Principal interface
getName method, 553
Principals (logins), 551
Print dialog box, 750
displaying page ranges in, 751, 759
native, 751, 755
print method
of *DocPrintJob*, 772
of JTable, 594, 596
of *Printable*, 749, 758–759
of PrintWriter, 69–70, 809–810
Print services, 769–772
document flavors for, 769–770
for images, 771
stream, 772–775
print, println functions (JavaScript), 448
print/PrintComponent.java, 756
print/PrintTestFrame.java, 755
Printable interface
implementing, 749, 753
objects, printing, 769
print method, 749, 758–759
Printer graphics context, 760
PrinterException, 750
PrinterJob class
defaultPage method, 758
getPrinterJob method, 749, 758
pageDialog method, 753, 755, 758
print method, 750–751, 758
printDialog method, 750–751, 758
setPageable method, 759
setPrintable method, 758

printf function (C), 793
printf method (PrintWriter), 69–70, 393
printf1/Printf1.c, 794
printf1/Printf1.java, 794
printf1/Printf1Test.java, 795
printf2/Printf2.c, 800
printf2/Printf2.java, 799
printf2/Printf2Test.java, 799
printf3/Printf3.c, 812
printf3/Printf3.java, 812
printf3/Printf3Test.java, 812
printf4/Printf4.c, 821
printf4/Printf4.java, 823
printf4/Printf4Test.java, 824
Printing
  clipped areas, 752
  counting pages during, 751
  images, 749–759
  layout of, 760
  multipage documents, 758–761
  number of copies for, 776
  page orientation of, 707, 753
  paper sizes in, 752
  quality of, 779
  selecting settings for, 750
  starting, 749
  text, 749–759
  using:
    banding for, 751
    transformations for, 761
Printing attributes, 776–784
  adding, 779
  categories of, 778–779
  checking values of, 779
  hierarchy of, 777
  retrieving, 779
PrintJob class (obsolete), 749
PrintJobAttribute interface, 776
  printing attributes of, 780–783
PrintJobAttributeSet interface, 777
println method
  of PrintWriter, 69–70, 72
  of System.out, 418–419
PrintQuality class, 779
PrintRequestAttribute interface, 776
  printing attributes of, 780–783
PrintRequestAttributeSet interface, 750, 777
PrintService interface
  createPrintJob method, 771–772
  getAttributes method, 784
  getName method, 771
printService/PrintServiceTest.java, 774
PrintServiceAttribute interface, 776
  printing attributes of, 780–783
PrintServiceAttributeSet interface, 777
PrintServiceLookup class
  lookupPrintServices method, 770–771
PrintStream class, 69
PrintWriter class, 68–70
  checkError method, 69–70
  constructor, 70
  print method, 69–70, 809–810
  printf method, 69–70, 393
  println method, 69–70, 72
Private keys, 566, 587
probeContentType method (Files), 118–119
processAnnotations method
    (ActionListenerInstaller), 470
Processing instructions (XML), 165
Processing tools, 467
Processor interface, 488
Programmer's Day, 367
Programs. *See* Applications
Project Panama, 846–848
Properties class, 160
Property files, 160
  character encoding of, 423
  event handlers in, 451
  for resources bundles, 422–423
  for string resources, 421
  for strings, 423
  no passwords in, 288
@Property annotation, 492
provides keyword, 528
Proxy objects, 470
PUBLIC identifier (DTD), 214
Public key ciphers, 566–573, 587–590
  performance of, 587
Public Key Cryptography Standard (PKCS)
    #5, 580
Pull parsers, 208
PushbackInputStream class, 65
  constructor, 67
  unread method, 67
put method
  of *Bindings*, 447
  of ByteBuffer, 137–138
  of CharBuffer, 139

of *ScriptEngine*, 447
of ScriptEngineManager, 447
PUT method (HttpRequest.Builder), 286
putClientProperty method (JComboBox), 636, 641
putNextEntry method (ZipOutputStream), 86–87
put*Xxx* methods (ByteBuffer), 133, 138

Q

\Q, in regular expressions, 145
QBE (query by example) tools, 297–298
QuadCurve2D class, 674, 680
QuadCurve2D.Double class, 675, 677, 690
QuadCurve2D.Float class, 675, 677
Quadratic curves, 679–680
quadTo method (Path2D.Float), 680, 691
Qualified exports, 525
Qualified names (XML), 202
Queries (databases), 298–300
by example, 297–298
executing, 308, 319–331
multiple, 311
populating row sets with results of, 340
preparing, 319–326
returning multiple results, 329–330
query/QueryTest.java, 321
&quot;, entity reference, 164
quoteReplacement method (Matcher), 157

R

R programming language, 444, 451
\r line feed character, 69, 170, 418
in e-mails, 287
\r, \R, in regular expressions, 145, 148
Race conditions, 49
Random class, 581
methods of, 45
Random numbers, streams of, 5, 13, 45, 48, 52
Random-access files, 81–85
randomAccess/RandomAccessTest.java, 83
RandomAccessFile class, 81–85, 131
constructor, 85
getChannel method, 136
getFilePointer method, 81, 85
length method, 81, 85
seek method, 81, 85
RandomGenerator class
methods of, 48
Randomness, 581
range, rangeClosed methods (*Xxx*Stream), 44, 46–47
Ranges, converting to streams, 50
Raster class
getDataElements method, 735, 738
getPixel, getPixels methods, 734, 739
Raster images, 721–749
constructing from pixels, 733–740
filtering, 740–749
readers/writers for, 722–732
rasterImage/RasterImageFrame.java, 736
read method
of CipherInputStream, 586
of FileInputStream, 56
of ImageIO, 722, 729
of ImageReader, 731
of InputStream, 56–58
of *Readable*, 62–63
of Reader, 60
read/config.dtd, 193
read/config.xml, 192
read/config.xsd, 193
read/XMLReadTest.java, 189
Readable interface, 61
read method, 62–63
ReadableByteChannel interface, 253
readAllBytes method (InputStream), 56, 58
readAll*Xxx* methods (Files), 118–119
readAttributes method (Files), 123–124
readBoolean method (*DataInput*), 79–80
readChar method (*DataInput*), 79–81
readDouble method (*DataInput*), 79–80, 90, 102
Reader class, 56, 61
read method, 60
READER class (DocFlavor), 770
readExternal method (*Externalizable*), 103–104
readFixedString method (DataIO), 82
readFloat method (*DataInput*), 79–80
readFully method (*DataInput*), 80
readInt method (*DataInput*), 79–81, 90
readLine method (Console), 71
readLong method (*DataInput*), 79–80
readNBytes method (InputStream), 56, 58
readObject method
of HashSet, 102
of ObjectInputStream, 89, 94, 102
ReadOnlyBufferException, 132
readResolve method (*Serializable*), 105–107

readShort method (*DataInput*), 79–80
readString method (Files), 118–119
readThumbnail method (ImageReader), 731
readUTF method (*DataInput*), 79–80
REAL data type (SQL), 301, 357
Receiver parameters, 482
Records, 72–75
  customizing serialization of, 104
  deserializing, 113
  fixed-size, 82–83
  ignoring serialVersionUID fields, 110
Rectangle2D class, 674–675
Rectangle2D.Double, Rectangle2D.Float classes, 675
RectangularShape class, 675
Redirects, of URLs, 274–275
reduce method (*Stream*), 41–43
reducing method (Collectors), 38
Reductions, 16, 41–43
ref attribute (XML Schema), 186
Reflection
  accessing:
    private members, 515–518, 522
    protected methods, 537
  constructing class trees, 656
  enumerating fields from a variable, 665
regex/RegexTest.java, 153
regexFilter method (RowFilter), 607, 616
Registry editor, 831, 836
Registry keys, 832–834
Regular expressions, 143–158
  escapes in, 73
  filtering, 607
  finding matches of, 149
  flags for, 155–158
  groups in, 151–154
  in DTDs, 179–180
  splitting sequences with, 6
relative method (*ResultSet*), 333, 337
Relativization, in URLs, 261
relativize method (*Path*), 116–117
releaseSavepoint method (*Connection*), 354, 356
ReleaseStringChars function (C), 798
ReleaseStringUTFChars function (C), 797–799
Release*Xxx*ArrayElements functions (C), 816–817, 819
reload method (DefaultTreeModel), 643, 649
remaining method (Buffer), 140–141
remove method (*AttributeSet*), 784
removeCellEditorListener method (*CellEditor*), 630
removeColumn method (JTable), 608, 614
removeNodeFromParent method (DefaultTreeModel), 643, 649
removeTreeModelListener method (*TreeModel*), 663, 671
RenderableImage interface, 769
Rendering (Swing)
  cells, 617–619
  columns, 600
  headers, 619
  nodes, 651–654
Rendering pipeline (AWT), 672–674
Renjin project, 444, 452
@Repeatable annotation, 483
replaceAll method
  of Matcher, 154, 157–158
  of String, 155
replaceFirst method (Matcher), 157
  of Matcher, 155
required keyword, 551
#REQUIRED attribute (DTD), 181
requires keyword, 508, 510, 512–513, 519, 523–525
requisite keyword, 551
RescaleOp class, 740, 747
Rescaling operation, 741
reset method
  of Buffer, 140–141
  of InputStream, 59
  of MessageDigest, 566
resolve method (*Path*), 116–117
resolveEntity method (*EntityResolver*), 177, 183
resolveSibling method (*Path*), 116–117
Resolving
  classes, 534
  relative URLs, 261
Resource bundles, 421–426
  loading, 423–424
  locating, 422–423
  lookup tables for, 424
  naming, 424
  searching for, 424
Resource editors, 421
@Resource annotation, 359, 483
ResourceBundle class
  extending, 424–425

getBundle method, 422–425
getKeys method, 425–426
getObject method, 424–425
getString method, 423, 425
getStringArray method, 426
handleGetObject method, 425–426
ResourceBundle.Control class
getCandidateLocales method, 423
Resources
hierarchy of, 422
in JAR files, 517
@Resources annotation, 483
ResourceScope interface, 847
Response headers, 264–265
Response pages, 269
Result interface, 227–228, 358
Result sets (databases)
accessing columns in, 309
analyzing, 308
closing, 312
for multiple queries, 311
iterating over rows in, 331
metadata for, 344
numbering rows in, 333
order of rows in, 309
retrieving multiple, 329–330
scrollable, 331–333
updatable, 331, 334–338
results method (Matcher), 158
ResultSet interface, 338
absolute method, 333, 337
afterLast, beforeFirst methods, 333, 337
cancelRowUpdates method, 335, 338
close method, 311–312
concurrency values, 333–334, 336, 338
deleteRow method, 335, 337
findColumn method, 311
first method, 333, 337
getBlob, getClob methods, 326–327
getConcurrency method, 332, 334, 336
getDate, getDouble, getInt methods, 309, 311
getMetaData method, 344, 353
getObject method, 309, 311
getRow method, 333, 337
getString method, 309, 311
getType method, 332, 336
getWarnings method, 314
insertRow method, 335, 337
isAfterLast, isBeforeFirst methods, 333, 337
isClosed method, 311
isFirst, isLast methods, 333, 337
iteration protocol, 308
last method, 333, 337
moveTo*Xxx*Row methods, 335, 337
next method, 308, 310, 331
previous method, 332, 336
relative method, 333, 337
type values, 333–334, 336, 338
update*Xxx* methods, 311, 334–335, 337–338
ResultSetMetaData interface, 344
getColumn*Xxx* methods, 344, 353
Retention policies, 485
@Retention annotation, 469, 483, 485
retire/Retire.java, 428
retire/RetireResources_de.java, 440
retire/RetireResources_zh.java, 440
retire/RetireResources.java, 439
retire/RetireStrings_de.properties, 441
retire/RetireStrings_zh.properties, 441
retire/RetireStrings.properties, 441
Retirement calculator example, 426–442
RETURN_GENERATED_KEYS field (*Statement*), 331
Return values, missing, 16
rewind method (Buffer), 140–141
RFC 821 standard, 287
RFC 822 standard, 380
RFC 1123 standard, 380
RFC 2396 standard, 261
RFC 2616 standard, 263
RFC 2911 standard, 783
RGB color model, 712, 734
Rhino engine, 444, 452
Rivest, Ronald, 564, 568, 587
Role-based authentication, 553
rollback method (*Connection*), 354–356
Root component (file system), 115
Root element (XML), 163
referencing schemas in, 185
Root node (Swing), 630–638
handles for, 637–638
separating children of, 637
rotate method (Graphics2D), 705, 709
Rotation, 704–705
interpolating pixels and, 741
with center point, 705
Round cap, 693
Round join, 694

Rounded rectangles, 677
RoundEnvironment interface, 489
RoundRectangle2D class, 674–675, 677
RoundRectangle2D.Double class, 675, 677, 690
RoundRectangle2D.Float class, 675, 677
Row sets (databases), 338–342
  cached, 339–344
  constructing, 339
  modifying, 339
  page size of, 340
RowFilter class, 606–608
  andFilter method, 607, 616
  dateFilter method, 607, 616
  include method, 607, 616
  notFilter method, 607, 616
  numberFilter method, 606, 616
  orFilter method, 607, 616
  regexFilter method, 607, 616
RowFilter.Entry class, 607
ROWID data type (SQL), 357–358
RowId interface, 358
Rows (databases), 295
  deleting/inserting, 335
  iterating through, 334
  order of, in result set, 309
  retrieving, 358
  selecting, 298
  updating, 334–338
Rows (Swing)
  filtering, 606–608
  height of, 603
  hiding, 608
  margins of, 603
  position, in a node, 643
  resizing, 603
  selecting, 593–594, 604
  sorting, 594, 605–606
RowSet interface, 338–341
  execute method, 341
  get*Xxx* methods, 341
  set*Xxx* methods, 340–341
RowSetFactory interface
  createCachedRowSet method, 339–340, 342
  create*Xxx*RowSet methods, 342
RowSetProvider class
  newFactory method, 339, 342
RSA algorithm, 568, 587
RSA Security, Inc., 580
rsa/RSATest.java, 588
rt.jar file, 530, 535
Ruby programming language, 444
run method (*Tool*), 457, 465
Runnable interface, 248
Runtime class
  exec method, 114
Runtime image file, 530
runtimeAnnotations/ActionListenerFor.java, 474
runtimeAnnotations/ActionListenerInstaller.java, 471

S

S (short), type code, 96, 806
\s, \S, in regular expressions, 146
@SafeVarargs annotation, 483
Sample values of pixels, 733
Sandbox, 577
Save points (databases), 354
Savepoint interface
  getSavepoint*Xxx* methods, 356
SAX (Simple API for XML) parser, 165, 203–208
  activating namespace processing in, 205
sax/SAXTest.java, 206
SAXParseException class
  get*Xxx*Number methods, 184
SAXParser class
  parse method, 204, 207
SAXParserFactory class
  is/setNamespaceAware methods, 207
  is/setValidating methods, 207
  newInstance, newSAXParser methods, 204, 207
  setFeature method, 205
SAXResult class, 228
SAXSource class, 227–228
  constructor, 233
Scalar functions, 328
scale method (Graphics2D), 704–705, 709
Scaling, 704–705
Scanner class, 70–72
  constructor, 116, 253
  findAll method, 158
  tokens method, 6, 10
  useLocale method, 393, 396
Scheduling applications
  computing dates for, 372–373
  time zones and, 367, 375
schema element (XML Schema), 187
Schemas, 352

Scheme programming language, 444
script/ScriptTest.java, 453
ScriptContext interface, 447
  get*Xxx*/set*Xxx* methods, 447–448
ScriptEngine interface
  createBindings method, 447
  eval method, 445–447
  get method, 447
  getContext method, 448
  put method, 447
ScriptEngineFactory interface
  getExtensions method, 445
  getMethodCallSyntax method, 449
  getMimeTypes method, 445
  getNames method, 445
ScriptEngineManager class, 444
  get method, 447
  getEngine*Xxx* methods, 444–445
  put method, 447
Scripting engines, 444–445
  adding variable bindings to, 446
  calling functions in, 448–450
  implementing Java interfaces, 449
  invoking, 445
Scripting languages, 444–456
Scripts, 444
  accessing classes in, 450
  compiling, 450–451
  executing, 446, 450
  invoking, 445
  redirecting I/O of, 447–448
  using Java method call syntax in, 449
Scroll panes (Swing)
  with tables, 593
  with trees, 644–645
scrollPathToVisible method (JTree), 644, 648
SecretKey interface, 581
SecretKeySpec class, 585
Secure random generator, 581
SecureRandom class
  setSeed method, 581
Security, 533–590
  bypassing constructors, 113–114
  bytecode verification, 545–549
  class loaders, 534–549
  code signing, 534
  different levels of, 562
  digital signatures, 562–577
  encryption, 578–590
  user authentication, 549–562
Security manager, 534
"Seek forward only" mode (ImageInputStream), 724
seek method (RandomAccessFile), 81, 85
select attribute (XSLT), 226
SELECT statement (SQL), 298–299
  executing, 308
  for LOBs, 326
  multiple, in a query, 329
  not supported in batch updates, 355
Selection models, 604
send method (HttpClient), 285
sendAsync method (HttpClient), 281, 285
separator constant (File), 64
Separators (file system), 64, 115
sequence element (XML Schema), 187
Sequences, producing, 5
Serial numbers, 90–91
serial/ObjectStreamTest.java, 93
@Serial annotation, 102–103
serialClone/SerialCloneTest.java, 110
SerialCloneable class, 110
Serializable interface, 89, 96, 487
  readResolve method, 105–107
  writeReplace method, 106–107
@Serializable annotation, 487
Serialization, 89–114
  cloning with, 110–113
  file format for, 94–101
  filters for, 114
  modifying default, 101–104
  of singletons, 104–107
  serial numbers for, 90–91
serialver program, 107
serialVersionUID constant, 107–110
server/EchoServer.java, 245
Servers
  accessing, 259–278
  connecting clients to, 238–240
  echo, 246–247
  implementing, 244–259
  invoking programs, 269
Server-side programs, 269–278
  redirecting URLs in, 274–275
ServerSocket class, 244–259
  accept method, 244, 247–248
  close method, 247
  constructor, 247

Service loading, 526–528
Service provider interfaces, 723
SERVICE_FORMATTED class (DocFlavor), 770
ServiceLoader class, 526
Servlets, 269, 461–467
Session class
  setDebug method, 288
set/Item.java, 497
set/SetTest.java, 498
setAllowsChildren method (DefaultMutableTreeNode), 639, 641
setAllowUserInteraction method (URLConnection), 262
setAsksAllowsChildren method (DefaultTreeModel), 639, 641
setAttribute, setAttributeNS methods (*Element*), 213–215
setAutoCommit method (*Connection*), 353, 355–356
setAutoCreateRowSorter method (JTable), 594, 596, 605
setAutoResizeMode method (JTable), 603, 613
setBinaryStream method (*Blob*), 327
SetBooleanArrayRegion function (C), 818
SetBooleanField function (C), 806
SetByteArrayRegion function (C), 818–819, 833
SetByteField function (C), 806
setCellEditor method (TableColumn), 620, 629
setCellRenderer method (TableColumn), 629
setCellSelectionEnabled method (JTable), 604, 614
setCharacterStream method (*Clob*), 327
SetCharArrayRegion function (C), 818–819
SetCharField function (C), 806
setClip method (Graphics), 710–711, 752
setClosedIcon method (DefaultTreeCellRenderer), 652, 654
setColumnSelectionAllowed method (JTable), 604, 614
setCommand method (*RowSet*), 340–341
setComparator method (DefaultRowSorter), 606, 615
setComposite method (Graphics2D), 673, 715, 721
setConnectTimeout method (URLConnection), 262, 268
setContentHandler method (*XMLReader*), 233
setContextClassLoader method (Thread), 537, 545
setCrc method (ZipEntry), 88
setCurrency method (NumberFormat), 402
setDataElements method (WritableRaster), 735, 739
setDate method (*PreparedStatement*), 320, 325
setDebug method (Session), 288
setDecomposition method (Collator), 413
setDefault method
  of CookieHandler, 274
  of Locale, 392, 394
setDefaultNamespace method (*XMLStreamWriter*), 221
setDefaultRenderer method (JTable), 618
setDoInput method (URLConnection), 262–263, 267
setDoOutput method (URLConnection), 262–263, 267, 272, 274
setDouble method (*PreparedStatement*), 320, 325
SetDoubleArrayRegion function (C), 818–819
SetDoubleField function (C), 802, 806
setEditable methodmethod (JTree), 645
setEntityResolver method (DocumentBuilder), 178, 183
setErrorHandler method (DocumentBuilder), 183
setErrorWriter method (*ScriptContext*), 448
setFeature method (SAXParserFactory), 205
setFillsViewportHeight method (JTable), 596
SetFloatArrayRegion function (C), 818–819
SetFloatField function (C), 806
setFrom method (MimeMessage), 288
setGroupingUsed method (NumberFormat), 399
setHeader*Xxx* methods (TableColumn), 619, 629
setIfModifiedSince method (URLConnection), 262–263, 268
setIgnoringElementContentWhitespace method (DocumentBuilderFactory), 182, 185
setInput method (ImageReader), 730
setInstanceFollowRedirects method (HttpURLConnection), 275
setInt method (*PreparedStatement*), 320, 325
SetIntArrayRegion function (C), 818–819
SetIntField function (C), 802, 806, 834
setLeafIcon method (DefaultTreeCellRenderer), 652, 654
setLevel method (ZipOutputStream), 87
setLocale method (MessageFormat), 415
setLogWriter method (DriverManager), 306
SetLongArrayRegion function (C), 818–819
SetLongField function (C), 806

setMaximum*Xxx*Digits methods (NumberFormat), 399
setMaxWidth method (TableColumn), 602, 615
setMethod method
  of ZipEntry, 88
  of ZipOutputStream, 87
setMinimum*Xxx*Digits methods (NumberFormat), 399
setMinWidth method (TableColumn), 602, 615
setName method (NameCallback), 561
setNamespaceAware method
  of DocumentBuilderFactory, 188, 201–202, 205, 213
  of SAXParserFactory, 207
SetObjectArrayElement function (C), 816, 819–820
SetObjectField function (C), 802, 806
setOpenIcon method (DefaultTreeCellRenderer), 652, 654
setOutput method (ImageWriter), 732
setOutputProperty method (Transformer), 214, 216
setPageable method (PrinterJob), 759
setPageSize method (*CachedRowSet*), 340, 342
setPaint method (Graphics2D), 672, 701–703
setParseIntegerOnly method (NumberFormat), 399
setPassword method
  of PasswordCallback, 561
  of *RowSet*, 340–341
setPixel, setPixels methods (WritableRaster), 733, 739
setPreferredWidth method (TableColumn), 602, 615
setPrefix method (*XMLStreamWriter*), 221
setPrintable method (PrinterJob), 758
setProperty method (XMLInputFactory), 209, 211
setReader method (*ScriptContext*), 447–448
setReadTimeout method (URLConnection), 262, 268
setRenderingHint, setRenderingHints methods (Graphics2D), 672
setRequestProperty method (URLConnection), 262–263, 268
setResizable method (TableColumn), 602, 615
setRootVisible method (JTree), 638, 640
setRowFilter method (DefaultRowSorter), 606–608, 615
setRowHeight, setRowMargin methods (JTable), 603, 613
setRowSelectionAllowed method (JTable), 604, 614
setRowSorter method (JTable), 605, 614
setSavepoint method (*Connection*), 356
setSeed method (SecureRandom), 581
setSelectionMode method (*ListSelectionModel*), 604, 615
SetShortArrayRegion function (C), 818–819
SetShortField function (C), 806
setShowsRootHandles method (JTree), 637, 640
setSize method (ZipEntry), 88
setSortable method (DefaultRowSorter), 605, 615
setSoTimeout method (Socket), 240–241
SetStatic*Xxx*Field functions (C), 805–806
setStrength method (Collator), 412
setString method
  of *PreparedStatement*, 320, 325
  of *RowSet*, 340
setStringConverter method (TableRowSorter), 615
setStroke method (Graphics2D), 672, 693, 701
setSubject method (MimeMessage), 288
setTableName method (*CachedRowSet*), 341–342
setText method (MimeMessage), 288
setTo*Xxx* methods (AffineTransform), 707, 709
setTransform method (Graphics2D), 707, 709
setURL method (*RowSet*), 340–341
setUseCaches method (URLConnection), 262
setUsername method (*RowSet*), 340–341
setUserObject method (*MutableTreeNode*), 632, 640
setValidating method
  of DocumentBuilderFactory, 182, 185
  of SAXParserFactory, 207
setValue method (Win32RegKey), 832–834
setValueAt method (*TableModel*), 600, 623
setWidth method (TableColumn), 602, 615
setWriter method (*ScriptContext*), 447–448
SGML (Standard Generalized Markup Language), 161
SHA-1 algorithm, 95, 563
Shamir, Adi, 568, 587
Shape interface, 675, 692
  implementing, 692
shape/ShapeTest.java, 682
ShapeMaker class
  getPointCount method, 681
  makeShape method, 681

ShapePanel class, 682
Shapes
  clipping, 672, 710–712
  combining, 673, 691–692
  control points of, 681–682
  drawing, 672–675
  filling, 672–673, 701–703
  rendering, 674
  transforming, 672
Shared libraries, 792, 830
Shear, 704–705
shear method (Graphics2D), 705, 709
Shift-JIS encoding, 77
short type
  printing, 69
  streams of, 43–49
  type code for, 96, 806
  vs. C types, 793
  writing in binary format, 78
ShortBuffer class, 139
ShortLookupTable class, 742, 748
shouldSelectCell method (*CellEditor*), 622–623, 630
shutdown*Xxx* methods (Socket), 252
Side files, 488
Signatures. *See* Digital signatures
Simple types, 185
SimpleDateFormat class, 414
SimpleDoc class, 771–772
SimpleFileVisitor class, 128
  visitFile method, 128–129
  visitFileFailed method, 128, 130
  *xxx*VisitDirectory methods, 130
SimpleJavaFileObject class, 459
  getCharContent method, 467
  openOutputStream method, 467
SimpleScriptContext class, 447
simpleType element (XML Schema), 186
Single value annotations, 477
Singletons, serializing, 104–107
size method
  of *BasicFileAttributes*, 124
  of Files, 123–124
skip method
  of InputStream, 58
  of *Stream*, 13
skipBytes method (*DataInput*), 80
skipNBytes method (InputStream), 58
SMALLINT data type (SQL), 301, 357
SMTP (Simple Mail Transport Protocol), 287–290
SOAP (Simple Object Access Protocol), 505
Socket class
  connect method, 241
  constructor, 240–241
  getInputStream method, 239–240, 244
  getOutputStream method, 240, 244
  isClosed, isConnected methods, 241
  is*Xxx*Shutdown methods, 252
  setSoTimeout method, 240–241
  shutdown*Xxx* methods, 252
socket/SocketTest.java, 239
SocketChannel class, 252
  open method, 252, 259
Sockets
  half-closing, 251–252
  interrupting, 252–259
  opening, 239
  server, 244–247
  timeouts of, 240–241
SocketTimeoutException, 241, 268
Software developer certificates, 577
sort method (Collections), 408
sorted method (*Stream*), 15–16
Source files
  character encoding of, 420–421
  reading from memory, 458
Source interface, 227, 358
Source-level annotations, 488–493
Space. *See* Whitespace
SPARC processor, big-endian order in, 79
split method
  of Pattern, 154, 156
  of String, 72, 154
Split packages, 514
splitAsStream method (Pattern), 6, 10, 154, 156
spliterator method (*Iterable*), 10
Spliterators class
  spliteratorUnknownSize method, 6, 9
SplittableRandom class, 45
  methods of, 52
sprint, sprintf functions (C), 798
SQL (Structured Query Language), 291, 295–301
  changing data inside databases, 300
  commands in, 303

data types in, 301, 357–358
equality testing in, 299
escapes in, 328–329
exceptions in, 312–314
executing statements in, 308–311
keywords in, 298
reading instructions from a file, 315
strings in, 299
vs. Java, 321
warnings in, 313
wildcards in, 299
SQLException class, 312–314, 332
getXxx methods, 312, 314
iterator method, 312, 314
rolling back and, 354
save points and, 356
SQLWarning class, 313, 332
getNextWarning method, 314
SQLXML data type (SQL), 357–358
Square cap, 693
Square root, computing, 22
SQuirreL program, 345
SRC, SRC_Xxx composition rules, 714–716
src.jar file, 826
sRGB standard, 734
Standard extensions, 535
Standard input, 56
StandardCharsets class, 77–78
StandardJavaFileManager interface, 457–459
getJavaFileObjectsFromXxx methods, 466
start method
of Matcher, 157
of MatchResult, 158
startDocument method (ContentHandler), 203, 207
startElement method (ContentHandler), 203–208
Stateless operations, 49
Statement interface, 308–311
addBatch method, 355–356
close, closeOnCompletion methods, 310, 312
execute method, 310, 316, 329, 331
executeBatch method, 355–356
executeLargeBatch method, 356
executeLargeUpdate method, 310
executeQuery method, 308, 310, 332, 334
executeUpdate method, 308, 310, 331, 354
getLargeUpdateCount method, 310
getMoreResults method, 329–330
getResultSet method, 310
getUpdateCount method, 310, 330
getWarnings method, 314
isClosed method, 310
RETURN_GENERATED_KEYS field, 331
using for multiple queries, 311
Statements (databases)
closing, 312
complex, 321
concurrently open, 311
executing, 308–311
grouping into transactions, 353–358
in batch updates, 355
multiple, 311
prepared, 319–326
truncations in, 313
static access modifier, 524
Static fields, in native code, 805–806
Static initialization blocks, 790
Static inner classes, 92
Static methods, calling from native code, 810–811
statusCode method (HttpResponse), 281, 286
StAX parser, 208–212, 217–222
namespace processing in, 209
no indented output in, 217–218
stax/StAXTest.java, 210
StAXSource class, 227
stopCellEditing method (CellEditor), 622–623, 630
Stored procedures, 328–329
Stream interface
allMatch, anyMatch methods, 17
collect method, 26–29, 43
concat method, 14
count method, 3–4, 16
distinct method, 15–16, 50
dropWhile method, 14
empty method, 5, 9
filter method, 3–11, 16
findAny method, 17
findFirst method, 16–17
flatMap method, 12
forEach method, 26, 29
forEachOrdered method, 26
generate method, 5, 9, 44
iterate method, 5, 9, 15, 44
iterator method, 26
limit method, 13, 50
map method, 11

mapToInt method, 42
max, min methods, 16–17
noneMatch method, 17
of method, 5, 9
ofNullable method, 6, 9, 24
peek method, 15–16
reduce method, 41–43
skip method, 13
sorted method, 15–16
takeWhile method, 14
toArray method, 26, 29
toList method, 6, 26, 29
unordered method, 50
stream method
of Arrays, 5, 9, 44
of Collection, 2–4
of Optional, 23–26
of StreamSupport, 6, 10
Streaming parsers, 165, 203–212
StreamPrintService class, 773
StreamPrintServiceFactory class, 772
getPrintService method, 773
lookupStreamPrintServiceFactories method, 773
StreamResult class, 216, 228
Streams, 1–53
collecting results from, 26–31
computing values from, 41–43
converting to/from arrays, 5, 26, 50
creating, 5–11
debugging, 15
empty, 5, 16, 41–42
encrypted, 585–586
filtering, 23
finite, 5
flattening, 12, 23
for print services, 772–775
infinite, 3, 5, 13, 15
intermediate operations for, 3
noninterference of, 7
of primitive types, 42–49
of random numbers, 45
parallel, 2, 17, 26, 32, 35, 38, 42, 49–53
processed lazily, 3, 7, 15
reductions of, 16
removing duplicates from, 15
returned by Files.lines, 51
sorted, 15, 50
splitting/combining, 13–14
summarizing, 27
terminal operations for, 3, 16
transformations of, 11–13, 44
vs. collections, 3
streams/CountLongWords.java, 3
streams/CreatingStreams.java, 7
streams/PrimitiveTypeStreams.java, 45
StreamSource class, 227
constructor, 233
transform method, 228
StreamSupport class
stream method, 6, 10
String class, 62
compareTo method, 407
format method, 393
replaceAll method, 155
split method, 72, 154
toLowerCase method, 11, 393
toUpperCase method, 393
trim method, 169, 396
STRING class (DocFlavor), 770
String parameters, 795–801
StringBuffer class, 62, 139
StringBuilder class, 62, 82
Strings
converting to code points, 11
encoding, 387, 420
fixed-size, I/O of, 82–83
in native code, 795–801
in SQL, 299
internationalizing, 421–423
ordering, 407
patterns for, 143–158
printing, 69
sorting, 408
splitting, 6
transforming to lower/uppercase, 11, 393
writing in binary format, 78
StringSource class, 458
Stroke interface, 693
stroke/StrokeTest.java, 696
StrokePanel class, 696
Strokes, 672, 693–701
dashed, 694–695
setting, 672
styling, 693–695
Stylesheets (XSLT), 223–234
Subject class
getPrincipals method, 553

Subjects (logins), 551
subSequence method (*CharSequence*), 63
subtract method (Area), 692
Subtraction operator, not associative, 42
Subtrees (Swing), 635, 652
  adding nodes to, 643
  collapsed and expanded, 635–636
Suetonius, Gaius Tranquillus, 540
sufficient keyword, 551
sum, summaryStatistics methods (primitive streams), 44, 46–47
summarizing*Xxx* methods (Collectors), 27, 30, 37
summing*Xxx* methods (Collectors), 36, 40
Sun, 291
SunJCE ciphers, 579
Superclasses
  adding, 110
  not serializable, 101
  type use annotations in, 480
Supplier interface
  get method, 11
@SupportedAnnotationTypes annotation, 488
SupportedValuesAttribute interface, 776
supportsBatchUpdates method (*DatabaseMetaData*), 355, 357
supportsResultSet*Xxx* methods (*DatabaseMetaData*), 332, 338
@SuppressWarnings annotation, 109, 483–484
SVG (Scalable Vector Graphics), 222–223
Swing, 591–671
  generating dynamic code for, 461
  tables in, 591–630
  trees in, 630–671
Symmetric ciphers, 579–580
  performance of, 587
SyncProviderException interface, 341–342
System class
  console method, 419
  loadLibrary method, 790, 792
System class loader, 535
SYSTEM identifier (DTD), 177, 214
System.err object, 69, 457
System.in object, 56, 69
  character encoding and, 419
System.out object, 69, 457
  character encoding and, 419
  println method, 418–419

T
t literal (SQL), 328
\t, in regular expressions, 145
Table cell renderers, 600
Table models (Swing), 592, 596–600
  updating after cell editing, 623
table/TableTest.java, 594
TableCellEditor interface
  getTableCellEditorComponent method, 621–622–623, 629
  implementing, 621, 623
tableCellRender/ColorTableCellEditor.java, 627
tableCellRender/ColorTableCellRenderer.java, 626
tableCellRender/PlanetTableModel.java, 625
tableCellRender/TableCellRenderFrame.java, 624
TableCellRenderer interface
  getTableCellRendererComponent method, 617, 629
  implementing, 617
TableColumn class, 601–603, 608
  constructor, 615
  setCellEditor method, 620, 629
  setCellRenderer method, 629
  setHeader*Xxx* methods, 619, 629
  setResizable, set*Xxx*Width methods, 602, 615
  setWidth method, 602, 615
TableColumnModel interface, 601–602
  getColumn method, 601, 614
TableModel interface, 605
  getColumnClass method, 600, 613
  getColumnName method, 600
  getColumnType method, 620
  getValueAt method, 596–597, 600, 619
  get*Xxx*Count methods, 596–597, 600
  implementing, 596
  isCellEditable method, 600, 619
  setValueAt method, 600, 623
tableModel/InvestmentTable.java, 598
tableRowColumn/PlanetTableFrame.java, 609
TableRowSorter class, 605–606
  setStringConverter method, 615
Tables (databases), 295
  changing data in, 300
  creating, 300
  duplicating data in, 297
  inspecting, 297–298
  metadata for, 343
  multiple, selecting data from, 299
  removing, 305

Tables (Swing), 591–630
  asymmetric, 600
  cells in:
    editing, 619–620
    rendering, 617–619
    selecting, 604
  columns in:
    accessing, 601–602
    adding/hiding, 608
    naming, 597–598
    rearranging, 593
    rendering, 600
    resizing, 593–594, 602–603
    selecting, 604
  combo boxes in, 620
  constructing, 593, 597
  custom editors in, 621–623
  headers in, 593
    rendering, 619
  index values in, 604
  printing, 594
  relationship between classes of, 602
  rows in:
    filtering, 606–608
    hiding, 608
    margins of, 603
    resizing, 603
    selecting, 593–594, 604
    sorting, 594, 605–606
  scrolling, 593
  text fields in, 619
TableStringConverter class
  toString method, 606, 616
takeWhile method (*Stream*), 14
@Target annotation, 469, 483–485
TCP (Transmission Control Protocol), 240
teeing method (Collectors), 38
telnet, 235–238
  activating/connecting, 236
  windows communicating in, 248–249
template element (XSLT), 225
Temporal interface, 365
TemporalAdjuster interface, 372
TemporalAdjusters class, 372–373
  dayOfWeekInMonth method, 373
  firstDayOf*Xxx*, lastDayOf*Xxx* methods, 373
  lastInMonth method, 373
  next method, 373
  nextOrSame method, 372–373
  ofDateAdjuster method, 372
  previous, previousOrSame methods, 373
TemporalAmount interface, 365–366, 370–371
test/TestDB.java, 306
@Test annotation, 468
@TestCase, @TestCases annotations, 487
Text, 68
  encoding of, 75–78
  generating from XML files, 226–228
  output, 68–70
  printing, 749–759, 769
  reading, 70–72
  saving objects in, 72–75
  transmitting through sockets, 244–259
  vs. binary data, 68
Text fields (Swing), 619
Text files, encoding of, 418, 420
Text nodes (XML)
  constructing, 213
  retrieving from XML, 169
TextCallbackHandler class, 554
textFile/TextFileTest.java, 73
TextLayout class, 711
  constructor, 712
  get*Xxx* methods, 712
TextStyle enumeration, 405
TextSyntax class, 779
TexturePaint class, 702–703
this keyword, 802
  annotating, 481–482
Thread class
  get/setContextClassLoader methods, 537, 545
threaded/ThreadedEchoServer.java, 249
ThreadedEchoHandler class, 247–251
Threads
  blocking, 57, 252–259
  executing scripts in, 446
  Internet connections with, 247–251
  race conditions in, 49
  referencing class loaders in, 537
Throw, ThrowNew functions (C), 819–820, 824
Throwable class, 820
Thumbnails, 725
Time
  current, 362
  formatting, 379–384, 403–407
  instances of, 373
  literals for, 328
  local, 373–374

measuring, 363
parsing, 381
zoned, 375–379, 403
Time class, 384
valueOf method, 385
TIME data type (SQL), 301, 328, 357
Time of day service, 236
timeline/TimeLine.java, 364
Timeouts, 240–241
Timestamp class, 384
toInstant method, 385
valueOf method, 385
TIMESTAMP data type (SQL), 301, 328, 357
Timestamps, 379
using instants as, 363
TimeZone class
getTimeZone method, 385
toZoneId method, 385
toAbsolutePath method (*Path*), 116–117
toArray method
of *AttributeSet*, 784
of primitive streams, 44, 46–47
of *Stream*, 26
of streams, 29
toCollection method (Collectors), 27, 30
toConcurrentMap method (Collectors), 32, 34
toDays method (Duration), 363, 365–366
toDaysPart method (Duration), 366
toFile method (*Path*), 117
toFormat method (DateTimeFormatter), 381, 385
toHours method (Duration), 363, 365–366
toInstant method
of Date, 384–385
of FileTime, 385
of Timestamp, 385
of ZonedDateTime, 375, 379
tokens method (Scanner), 6, 10
toLanguageTag method (Locale), 391, 394
toList method
of Collectors, 26, 30
of Stream, 6, 26, 29
toLocal*Xxx* methods
of Local*Xxx*, 385
of ZonedDateTime, 379
toLowerCase method (String), 11, 393
toMap method (Collectors), 31–34
toMillis, toMinutes, toNanos methods (Duration), 363, 365–366
toNanoOfDay method (LocalTime), 374
Tool interface
run method, 457, 465
ToolProvider class
getSystemJavaCompiler method, 457
toPath method (File), 117–118
toSecondOfDay method (LocalTime), 374
toSeconds method (Duration), 363, 365–366
toSet method (Collectors), 26, 30, 36
toString method
implementing with annotations, 490–493
of *Annotation*, 476
of *CharSequence*, 63
of Currency, 403
of Locale, 394
of TableStringConverter, 606, 616
of Variable, 665
toUnmodifiableList method (Collectors), 30
toUnmodifiableMap method (Collectors), 34
toUnmodifiableSet method (Collectors), 30
toUpperCase method (String), 393
toZonedDateTime method (GregorianCalendar), 384–385
toZoneId method (TimeZone), 385
Transactions, 353–358
committing, 353
error handling in, 355
rolling back, 353
transferTo method (InputStream), 58
transform method
of Graphics2D, 673, 707, 709
of StreamSource, 228
of Transformer, 214, 216, 227
transform/makehtml.xsl, 229
transform/makeprop.xsl, 229
transform/TransformTest.java, 230
Transformations, 672, 703–709
affine, 706, 740
composing, 705–706
fundamental types of, 704–705
matrices for, 706
order of, 705
setting, 673
using for printing, 761
Transformer class
setOutputProperty method, 214, 216
transform method, 214, 216, 227
TransformerFactory class
newInstance method, 214, 216
newTransformer method, 214, 216, 232

transient keyword, 101–102
transitive keyword, 523–524
translate method (Graphics2D), 705, 709, 761
Translation, 704–705
Transparency, 712–721
Traversal order, 649–650
Tree events, 654–662
Tree models
  constructing, 632, 663
  custom, 662–671
  default, 632
Tree parsers, 165
Tree paths, 641–645
  constructing, 644, 650
Tree selection listeners, 655
tree/SimpleTreeFrame.java, 634
TreeCellRenderer interface, 651–654
  getTreeCellRendererComponent method, 652–654
  implementing, 652
treeEdit/TreeEditFrame.java, 645
TreeMap class, 32
TreeModel interface, 632, 642
  add/removeTreeModelListener methods, 663, 671
  getChild, getChildCount methods, 663–665, 670
  getIndexOfChild method, 663, 670
  getRoot method, 663–665, 670
  implementing, 632
  isLeaf method, 640, 663, 671
  valueForPathChanged method, 664, 671
treeModel/ObjectInspectorFrame.java, 666
treeModel/ObjectTreeModel.java, 667
treeModel/Variable.java, 669
TreeModelEvent class, 671
TreeModelListener interface, 663
  treeNodes*Xxx* methods, 663, 671
  treeStructureChanged method, 663, 671
TreeNode interface, 632, 642
  children method, 649
  getAllowsChildren method, 640
  getChildAt method, 648
  getChildCount method, 649
  getParent method, 648, 650
  isLeaf method, 639–640
TreePath class, 642
  getLastPathComponent method, 642, 648
treeRender/ClassNameTreeCellRenderer.java, 661
treeRender/ClassTreeFrame.java, 657
Trees (Swing), 630–671
  adding listeners to, 655
  background color for, 652
  connecting lines in, 636–637
  displaying, 632–649
  editing, 641–645, 664
  handles in, 635, 637, 652
  hierarchy of classes for, 633
  indexes in, 643
  infinite, 666
  leaves in, 630, 638–639, 651, 663
  nodes in, 630, 639, 651, 663
  paired with other components, 654
  rendering, 651–654
  scrolling to newly added nodes, 644–645
  structure of, 630
  subtrees in, 635–636
  traversals for, 649–651
  updating vs. reloading, 643
  user objects for, 632, 643
  view of, 643
  with horizontal lines, 637
TreeSelectionEvent class
  getPath method, 662
  getPaths method, 656, 662
TreeSelectionListener interface
  implementing, 654–662
  valueChanged method, 655, 657, 662
TreeSelectionModel interface, 655
trim method (String), 169, 396
Troubleshooting. *See* Debugging
True Odds: How Risks Affect Your Everyday Life (Walsh), 563
tryLock method (FileChannel), 141–143
try-with-resources statement, 62
  closing files with, 124, 126
  for database connections, 312
  with locks, 142
ts literal (SQL), 328
Type bounds, type use annotations in, 480
Type codes, 96, 806
Type definitions, 185
  anonymous, 187
  nesting, 186
type method (*XPathEvaluationResult*), 199
Type parameters, annotating, 479
Type use annotations, 480

TYPE_BICUBIC, TYPE_BILINEAR fields (AffineTransformOp), 741, 747
TYPE_BYTE_GRAY field (BufferedImage), 736, 738
TYPE_BYTE_INDEXED field (BufferedImage), 738
TYPE_INT_ARGB field (BufferedImage), 733–734, 738
TYPE_NEAREST_NEIGHBOR field (AffineTransformOp), 741, 747
TypeElement interface, 489
Types. *See* Data types
Typesafe enumerations, 104–107

U

U.S. government on exporting encryption methods, 540
\u, in regular expressions, 145
UDP (User Datagram Protocol), 240
UIManager class, 618
Unicode standard, 44, 75
  character order in, 407
  converting to binary data, 68
  in property files, 423
  input/output streams and, 56
  native code and, 795
  normalization forms in, 409
  using for all strings, 387
Units of measurement, 164
UNIX operating system
  authentication in, 550
  line feed in, 69, 418
  paths in, 115
  specifying locales in, 392
UnixLoginModule class, 551
UnixNumericGroupPrincipal class, 551
UnixPrincipal class, 551
UnknownHostException, 239
unordered method (*BaseStream*), 50, 53
Unparsed external entities, 181
unread method (PushbackInputStream), 67
UnsatisfiedLinkError, 787
until method (LocalDate), 367, 370
update method
  of Cipher, 579, 582, 585–586
  of MessageDigest, 564, 566
UPDATE statement (SQL), 300, 320, 334
  executing, 308, 310, 326
  in batch updates, 355
  truncations in, 313
  vs. methods of *ResultSet*, 335
update*Xxx* methods (*ResultSet*), 311, 334–335, 337–338
URI class, 280
  get*Xxx* methods, 261
  no resource accessing with, 260
uri method (HttpRequest.Builder), 279–281, 286
URIs (Uniform Resource Identifiers), 200, 260
  absolute vs. relative, 260–261
  base, 261
  hierarchical, 260
  namespace, 199–202
  opaque vs. nonopaque, 260
  schemes for, 260
  with HTTP, 280
URISyntax class, 779
URL class (DocFlavor), 770
URL class (java.lang.Object), 259–261, 280
  accepted schemes for, 260
  openConnection method, 262, 267
  openStream method, 64, 259, 267
URLClassLoader class
  addURLs method, 537
  constructor, 545
  getURLs method, 537
  loadClass method, 536
URLConnection class, 259, 262–269, 279
  connect method, 262, 264, 268
  getConnectTimeout method, 268
  getContent method, 269
  getContentEncoding, getContentType methods, 262, 265, 269, 274
  getContentLength method, 262, 265, 268
  getDate method, 262, 265, 269
  getDoInput, getDoOutput methods, 267
  getExpiration method, 262, 265, 269
  getHeader*Xxx* methods, 262–265, 268
  getIfModifiedSince method, 268
  getInputStream method, 262, 269, 272, 274
  getLastModified method, 262, 265, 269
  getOutputStream method, 262, 269, 272
  getReadTimeout method, 268
  getRequestProperty method, 268
  setAllowUserInteraction method, 262
  setConnectTimeout method, 262, 268
  setDoInput method, 262–263, 267
  setDoOutput method, 262–263, 267, 272, 274

setIfModifiedSince method, 262–263, 268
setReadTimeout method, 262, 268
setRequestProperty method, 262–263, 268
setUseCaches method, 262
urlConnection/URLConnectionTest.java, 265
URLDecoder class
decode method, 278
URLEncoder class
encode method, 278
URLs (Uniform Resource Locators), 260
attaching parameters to, 270
connections via, 259
encoding, 271
for databases, 302
for namespace identifiers, 200
redirecting, 274–275
relative vs. absolute, 177
URNs (Uniform Resource Names), 260
US Letter paper, 752
useLocale method (Scanner), 393, 396
User coordinates, 704
User objects, 632
User-Agent request parameter, 275
Users
authentication of, 549–562
preferences of, 141
uses keyword, 527–528
UTC (Coordinated Universal Time), 375
UTF-8 encoding, 76–79
byte order in, 76, 420
for text files, 418
modified, 79–80, 420, 795–798
UTF-16 encoding, 44, 68, 76, 79
byte order in, 76
in regular expressions, 145
native code and, 795

V

V (void), type code, 806
\v, \V, in regular expressions, 146
validateObject method (*ObjectInputValidation*), 113–114
Validation, 175–194
activating, 182
adding to classes, 101
value method (*XPathEvaluationResult*), 196, 199
valueChanged method (*TreeSelectionListener*), 655, 657, 662
valueForPathChanged method (*TreeModel*), 664, 671
value-of element (XSLT), 226
valueOf method (date/time legacy classes), 385
VARCHAR data type (SQL), 301, 357
VarHandle class, 518
Variable class, 664
toString method, 665
Variable handles, 518
VariableElement interface, 489
Variables
annotating, 468, 480
binding, 446
fields of, 665
initializing, 546
scope of, 447
Vendor name, of a reader, 723
verifier/VerifierTest.java, 549
Verifiers, 545–549
Version number, of a reader, 723
Versioning, 107–110
view/ViewDB.java, 345
visitFile, visitFileFailed methods
of *FileVisitor*, 127
of SimpleFileVisitor, 128–130

W

\w, \W, in regular expressions, 146
W3C (World Wide Web Consortium), 166, 204
walk method (Files), 125–126
walkFileTree method (Files), 127–129
warning method (ErrorHandler), 183–184
Warnings
SQLWarning, 313–314
suppressing, 484
WBMP format, 722
WeakReference object, 666
Web applications, 358–360
Web crawlers, 204
with SAX parser, 205
with StAX parser, 209
Web pages
dynamic, 461–467
separating class loaders for, 539
WebRowSet interface, 339
Weekends, 368
Weeks, 405

WHERE statement (SQL), 299
Whitespace
  ignoring, while parsing, 168–169
  in e-mail URIs, 271
  in regular expressions, 146
Wildcards, type use annotations in, 480
Wilde, Oscar, 388
win32reg/Win32RegKey.c, 838
win32reg/Win32RegKey.java, 836
win32reg/Win32RegKeyTest.java, 845
Win32RegKey class, 832, 835
  getValue method, 832–833
  names method, 832
  setValue method, 832–834
Win32RegKeyNameEnumeration class, 834–835
Windows operating system
  activating telnet in, 236
  authentication in, 550
  character encodings in, 418
  classpath in, 302
  compiling invocation API, 829
  dynamic linking in, 826
  glob syntax in, 127
  line feed in, 69, 418
  look-and-feel of, 636
  paths in, 64, 115
  registry, accessing from native code, 830–846
  resources in, 421
  using Microsoft compiler, 789–790
Windows-1252 encoding, 418–419
with method (*Temporal*), 372–373
withLocale method (DateTimeFormatter), 380, 384, 404, 407
with*Xxx* methods
  of LocalDate, 370
  of LocalTime, 374
  of Period, 371
  of ZonedDateTime, 378
Words, in regular expressions, 146
Working directory, 64
wrap method (ByteBuffer), 138, 140
WritableByteChannel interface, 253
WritableRaster class, 733
  setDataElements method, 735, 739
  setPixel, setPixels methods, 733, 739
write method
  of CipherOutputStream, 586
  of Files, 118–119
  of ImageIO, 722, 729
  of ImageWriter, 725, 732
  of OutputStream, 57, 59
  of Writer, 60
write/XMLWriteTest.java, 218
writeAttribute method (*XMLStreamWriter*), 217, 222
writeBoolean method (DataOutput), 78, 80
writeByte method (DataOutput), 78, 80
writeCData method (*XMLStreamWriter*), 222
writeChar method (DataOutput), 78, 80–81
writeCharacters method (*XMLStreamWriter*), 217, 222
writeChars method (DataOutput), 78–80
writeComment method (*XMLStreamWriter*), 222
writeDouble method (DataOutput), 78, 80, 90, 102
writeDTD method (*XMLStreamWriter*), 222
writeEmptyElement method (*XMLStreamWriter*), 217, 221
writeEnd*Xxx* methods (*XMLStreamWriter*), 217, 221
writeExternal method (*Externalizable*), 103–104
writeFixedString method (DataIO), 82
writeFloat method (DataOutput), 78, 80
writeInsert method (ImageWriter), 725, 732
writeInt method (DataOutput), 78, 80–81, 90
writeLong method (DataOutput), 78, 80
writeObject method
  of HashSet, 102
  of ObjectOutputStream, 89, 94, 102
Writer class, 56, 61–62
  write method, 60
writeReplace method (*Serializable*), 106–107
writeShort method (DataOutput), 78, 80
writeStart*Xxx* methods (*XMLStreamWriter*), 217, 221
writeString method (Files), 118–119
writeUTF method (DataOutput), 78–80

X

X.509 format, 570
\x, in regular expressions, 145
XHTML (Extensible Hypertext Markup Language), 162, 204
XML (Extensible Markup Language), 159–234
  annotated version of the standard, 161
  attributes in, 162

case sensitivity of, 162
end and empty tags in, 162
in databases, 358
namespaces in, 199–202
vs. HTML, 161–162
XML catalogs, 177
XML documents
DTDs in, 162, 176–185
format of, 161
generating, 212–223
from non-XML legacy data, 228
HTML files from, 223–226
plain text from, 226–228
with StAX, 217–222
locating information in, 194–199
malformed, 217–218
parsing, 165–175
structure of, 162–166, 176
validating, 175–194
with/without namespaces, 213–214
XML Schema, 176, 185–188
attributes in, 187
documentation for, 200
parsing with, 188
referencing in XML documents, 185
repeated elements in, 187
type definitions in, 185–187
XML/JSON binding, 517
XMLInputFactory class
createXMLStreamReader method, 211
newInstance method, 211
setProperty method, 209, 211
xmlns attribute (XSLT), 201
XMLOutputFactory class
createXMLStreamWriter method, 217, 221
newInstance method, 217, 221
XMLReader interface
implementing, 227
parse method, 233
setContentHandler method, 233
XMLStreamReader interface
getAttribute*Xxx* methods, 209, 212
getName, getLocalName, getText methods, 212
hasNext method, 211
is*Xxx* methods, 212
next method, 212
XMLStreamWriter interface, 217
close method, 222
not autocloseable, 217
setDefaultNamespace, setPrefix methods, 221
writeAttribute method, 217, 222
writeCData method, 222
writeCharacters method, 217, 222
writeComment method, 222
writeDTD method, 222
writeEmptyElement method, 217, 221
writeEnd*Xxx* methods, 217, 221
writeStart*Xxx* methods, 217, 221
XOR composition rule, 714
XPath (XML Path Language), 194–199
count function, 195
XPath interface
evaluate method, 195–196, 199
evaluateExpression method, 195–196, 199
xpath/XPathTest.java, 197
XPathEvaluationResult interface
type method, 199
value method, 196, 199
XPathFactory class
newInstance method, 195, 199
newXpath method, 199
XPathNodes class, 196
xs:, xsd: prefixes (XSL Schema), 186
xsd:attribute element (XML Schema), 187
xsd:choice element (XML Schema), 187
xsd:complexType element (XML Schema), 186
xsd:element element (XML Schema), 186
xsd:enumeration element (XML Schema), 186
xsd:schema element (XML Schema), 187
xsd:sequence element (XML Schema), 187
xsd:simpleType element (XML Schema), 186
xsl:apply-templates element (XSLT), 225
xsl:output element (XSLT), 225
xsl:template element (XSLT), 225
xsl:value-of element (XSLT), 226
XSLT (Extensible Stylesheet Language Transformations), 214, 223–234
copying attribute values in, 226
templates in, 225
XSLT processors, 223

Y

Yasson framework, 518
Year, YearMonth classes, 368

Z

Z (boolean), type code, 96, 806
\z, \Z, in regular expressions, 148

ZIP archives, 85–88
  for JMOD files, 530
  reading, 66, 85–86
  writing, 86
Zip code lookup, 272
ZIP file systems, 130–131
ZipEntry class
  constructor, 87
  get*Xxx*, set*Xxx* methods, 88
  isDirectory method, 88
ZipFile class
  constructor, 88
  entries method, 88
  get*Xxx* methods, 88
ZipInputStream class, 60, 85
  closeEntry method, 85–87
  constructor, 87
  getNextEntry method, 85–87
  read method, 85
ZipOutputStream class, 60, 86
  closeEntry method, 86–87
  constructor, 87
  putNextEntry method, 86–87
  setLevel, setMethod methods, 87
ZonedDateTime class, 375–379
  format method, 404
  from method, 384–385
  get*Xxx* methods, 378–379
  isAfter, isBefore methods, 379
  legacy classes and, 384–385
  minus, minus*Xxx* methods, 378
  now method, 378
  of, ofInstant methods, 375, 378
  parse method, 384, 404, 407
  plus, plus*Xxx* methods, 378
  toInstant method, 375, 379
  toLocal*Xxx* methods, 379
  with*Xxx* methods, 378
zonedtimes/ZonedTimes.java, 377
ZoneId class, 385
  getAvailableZoneIds method, 375
  of method, 375