

CORE JAVA

Volume II: Advanced Features

FOURTEENTH EDITION

ORACLE

Cay S. Horstmann

FREE SAMPLE CHAPTER

This page intentionally left blank

Core Java

Volume II: Advanced Features

Fourteenth Edition

Cay S. Horstmann

Pearson

Cover image: surangaw/stock.adobe.com
Figures 3.3, 4.8: Mozilla Foundation
Figures 4.1-4.5, 5.3, 5.4: Microsoft Corporation
Figure 4.6: USPS
Figures 5.6, 7.1-7.6, 9.7, 10.1, 10.3, 10.8, 10.10, 10.11, 10.14-10.16, 11.4, 11.5, 11.8-11.27, 11.29-11.34, 12.1-12.5, 12.7-12.10, 12.14-12.26, 12.28-12.30, 12.39, 12.44-12.46, 12.54-12.56, 12.60, 12.62: Oracle Corporation
Figure 9.3: HHD Software Ltd
Figure 12.51: Shao-Chun Wang/123RF
Figures 12.57-12.59: Pearson Education

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

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.

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

Author websites are not owned or managed by Pearson.

Visit us on the Web: informit.com

Library of Congress Control Number: 2025945104

Copyright © 2026 Pearson Education, Inc.
Hoboken, New Jersey

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.

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

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 pearson.com/en-us/global-permission-granting.html.

ISBN-13: 978-0-13-555869-0
ISBN-10: 0-13-555869-7

Contents

Preface xi
Acknowledgments xv
1. Streams 1
1.1. From Iterating to Stream Operations 1
1.2. Stream Creation 3
1.3. The `filter`, `map`, and `flatMap` Methods 9
1.4. Extracting Substreams and Combining Streams 11
1.5. Other Stream Transformations 12
1.6. Simple Reductions 13
1.7. The Optional Type 14
1.7.1. Getting an Optional Value 14
1.7.2. Consuming an Optional Value 15
1.7.3. Pipelining Optional Values 16
1.7.4. How Not to Work with Optional Values 17
1.7.5. Creating Optional Values 18
1.7.6. Composing Optional Value Functions with `flatMap` 18
1.7.7. Turning an Optional into a Stream 19
1.8. Collecting Results 21
1.9. Collectors 22
1.9.1. Collecting into Collections 25
1.9.2. Collecting into Maps 27
1.9.3. Grouping and Partitioning 30
1.9.4. Downstream Collectors 31
1.9.5. Implementing Collectors 35
1.10. Reduction Operations 38
1.11. Gatherers 40
1.11.1. Predefined Gatherers 40
1.11.2. Implementing Gatherers 42
1.12. Primitive Type Streams 46
1.13. Parallel Streams 51
2. Input and Output 57
2.1. Input/Output Streams 57
2.1.1. Reading and Writing Bytes 57
2.1.2. The Complete Stream Zoo 60
2.1.3. Combining Input/Output Stream Filters 64
2.1.4. Text Input and Output 67
2.1.5. How to Read Text Input 68
2.1.6. How to Write Text Output 69
2.1.7. Saving Objects in Text Format 71
2.1.8. Character Encodings 74
2.1.9. Default Character Encodings 76
2.1.10. Reading Character Input 77

2.2. Reading and Writing Binary Data ... 78
2.2.1. The DataInput and DataOutput Interfaces ... 78
2.2.2. Random-Access Files ... 80
2.2.3. ZIP Archives ... 84
2.3. Working with Files ... 86
2.3.1. Paths ... 86
2.3.2. Reading and Writing Files ... 89
2.3.3. Creating Files and Directories ... 90
2.3.4. Copying, Moving, and Deleting Files ... 91
2.3.5. Getting File Information ... 93
2.3.6. Visiting Directory Entries ... 95
2.3.7. Glob Patterns ... 97
2.3.8. Directory Streams ... 98
2.3.9. ZIP File Systems ... 100
2.4. Memory-Mapped Files ... 101
2.4.1. Memory-Mapped File Performance ... 101
2.4.2. The Buffer Data Structure ... 107
2.5. File Locking ... 110
2.6. Object Input/Output Streams and Serialization ... 113
2.6.1. Saving and Loading Serializable Objects ... 113
2.6.2. Understanding the Object Serialization File Format ... 118
2.6.3. Transient Fields ... 125
2.6.4. The readObject and writeObject Methods ... 125
2.6.5. The readExternal and writeExternal Methods ... 127
2.6.6. The readResolve and writeReplace Methods ... 128
2.6.7. Versioning ... 129
2.6.8. Deserialization and Security ... 132
2.6.9. Using Serialization for Cloning ... 133
2.7. Regular Expressions ... 134
2.7.1. The Regular Expression Syntax ... 135
2.7.2. Testing a Match ... 140
2.7.3. Finding All Matches in a String ... 140
2.7.4. Groups ... 142
2.7.5. Splitting along Delimiters ... 144
2.7.6. Replacing Matches ... 145
2.7.7. Flags ... 146
3. XML ... 149
3.1. Introducing XML ... 149
3.2. The Structure of an XML Document ... 151
3.3. Parsing an XML Document ... 154
3.4. Validating XML Documents ... 162
3.4.1. Document Type Definitions ... 163
3.4.2. XML Schema ... 170
3.4.3. A Practical Example ... 173
3.5. Locating Information with XPath ... 178
3.6. Using Namespaces ... 182
3.7. Streaming Parsers ... 185
3.7.1. Using the SAX Parser ... 185
3.7.2. Using the StAX Parser ... 189
3.8. Generating XML Documents ... 192
3.8.1. Documents without Namespaces ... 192

3.8.2. Documents with Namespaces ... 193
3.8.3. Writing Documents ... 193
3.8.4. Writing an XML Document with StAX ... 196
3.9. XSL Transformations ... 201
4. Networking ... 211
4.1. Connecting to a Server ... 211
4.1.1. Using Telnet ... 211
4.1.2. Connecting to a Server with Java ... 213
4.1.3. Socket Timeouts ... 215
4.1.4. Internet Addresses ... 216
4.2. Implementing Servers ... 218
4.2.1. Server Sockets ... 218
4.2.2. Serving Multiple Clients ... 221
4.2.3. Half-Close ... 223
4.2.4. Interruptible Sockets ... 224
4.2.5. Secure Socket Communication ... 227
4.3. Getting Web Data ... 231
4.3.1. URLs and URIs ... 231
4.3.2. Using a `URLConnection` to Retrieve Information ... 233
4.3.3. Posting Form Data ... 239
4.4. The HTTP Client ... 247
4.4.1. The `HttpClient` Class ... 247
4.4.2. The `HttpRequest` class and Body Publishers ... 247
4.4.3. The `HttpResponse` Interface and Body Handlers ... 248
4.4.4. Asynchronous Processing ... 249
4.5. The Simple HTTP Server ... 254
4.5.1. The Command-Line Tool ... 255
4.5.2. The HTTP Server API ... 255
4.5.3. Handlers ... 256
4.5.4. Filters ... 257
4.6. Sending E-Mail ... 259
5. Database Programming ... 263
5.1. The Design of JDBC ... 263
5.1.1. JDBC Driver Types ... 264
5.1.2. Typical Uses of JDBC ... 265
5.2. The Structured Query Language ... 265
5.3. JDBC Configuration ... 271
5.3.1. Database URLs ... 272
5.3.2. Driver JAR Files ... 272
5.3.3. Starting the Database ... 272
5.3.4. Connecting to the Database ... 273
5.4. Working with JDBC Statements ... 276
5.4.1. Executing SQL Statements ... 276
5.4.2. Managing Connections, Statements, and Result Sets ... 278
5.4.3. Analyzing SQL Exceptions ... 279
5.4.4. Populating a Database ... 281
5.5. Query Execution ... 284
5.5.1. Prepared Statements ... 284
5.5.2. Reading and Writing LOBs ... 290
5.5.3. SQL Escapes ... 291
5.5.4. Multiple Results ... 293

5.5.5. Retrieving Autogenerated Keys ... 294
5.6. Scrollable and Updatable Result Sets ... 294
5.6.1. Scrollable Result Sets ... 294
5.6.2. Updatable Result Sets ... 296
5.7. Row Sets ... 300
5.7.1. Constructing Row Sets ... 300
5.7.2. Cached Row Sets ... 300
5.8. Metadata ... 303
5.9. Transactions ... 310
5.9.1. Programming Transactions with JDBC ... 310
5.9.2. Save Points ... 311
5.9.3. Batch Updates ... 311
5.9.4. Advanced SQL Types ... 313
5.10. Connection Management in Web and Enterprise Applications ... 314
6. The Date and Time API ... 317
6.1. The Time Line ... 317
6.2. Local Dates ... 321
6.3. Date Adjusters ... 325
6.4. Local Time ... 326
6.5. Zoned Time ... 327
6.6. Formatting and Parsing ... 331
6.7. Interoperating with Legacy Code ... 336
7. Internationalization ... 339
7.1. Locales ... 339
7.1.1. Why Locales? ... 340
7.1.2. Specifying Locales ... 340
7.1.3. The Default Locale ... 343
7.1.4. Display Names ... 344
7.2. Number Formats ... 346
7.2.1. Formatting Numeric Values ... 346
7.2.2. The `DecimalFormat` Class ... 349
7.2.3. Currencies ... 352
7.3. Date and Time ... 353
7.4. Collation and Normalization ... 356
7.5. Message Formatting ... 361
7.5.1. Formatting Numbers and Dates ... 361
7.5.2. Choice Formats ... 363
7.6. Text Boundaries ... 366
7.7. Text Input and Output ... 366
7.7.1. Text Files ... 366
7.7.2. Line Endings ... 367
7.7.3. The Console ... 367
7.7.4. The UTF-8 Byte Order Mark ... 368
7.7.5. Character Encoding of Source Files ... 368
7.8. Resource Bundles ... 369
7.8.1. Locating Resource Bundles ... 369
7.8.2. Property Files ... 370
7.8.3. Bundle Classes ... 371
7.9. A Complete Example ... 373

8. Compiling and Scripting 379
8.1. The Compiler API 379
8.1.1. Invoking the Compiler 379
8.1.2. Launching a Compilation Task 380
8.1.3. Capturing Diagnostics 380
8.1.4. Reading Source Files from Memory 381
8.1.5. Writing Byte Codes to Memory 381
8.1.6. An Example: Dynamic Java Code Generation 383
8.2. Scripting for the Java Platform 387
8.2.1. Getting a Scripting Engine 388
8.2.2. Script Evaluation and Bindings 389
8.2.3. Redirecting Input and Output 391
8.2.4. Calling Scripting Functions and Methods 391
8.2.5. Compiling a Script 393
8.2.6. An Example: Script Sheets 394
9. Security 397
9.1. Class Loaders 398
9.1.1. The Class-Loading Process 398
9.1.2. The Class Loader Hierarchy 399
9.1.3. Using Class Loaders as Namespaces 401
9.1.4. Writing Your Own Class Loader 401
9.1.5. Bytecode Verification 405
9.2. User Authentication 409
9.2.1. The JAAS Framework 409
9.2.2. JAAS Login Modules 412
9.3. Digital Signatures 418
9.3.1. Message Digests 419
9.3.2. Password Hashing 421
9.3.3. Message Signing 422
9.3.4. Verifying a Signature 424
9.3.5. The Authentication Problem 427
9.3.6. Certificate Signing 429
9.3.7. Code Signing 430
9.4. Encryption 430
9.4.1. Symmetric Ciphers 431
9.4.2. Key Generation 433
9.4.3. Cipher Streams 437
9.4.4. Public Key Ciphers 438
10. Graphical User Interface Programming 443
10.1. A History of Java User Interface Toolkits 443
10.2. Displaying Frames 445
10.2.1. Creating a Frame 445
10.2.2. Frame Properties 447
10.3. Displaying Information in a Component 450
10.3.1. Working with 2D Shapes 454
10.3.2. Using Color 460
10.3.3. Using Fonts 461
10.3.4. Displaying Images 467
10.4. Event Handling 469
10.4.1. Basic Event Handling Concepts 469
10.4.2. Example: Handling a Button Click 470

10.4.3. Specifying Listeners Concisely 474
10.4.4. Adapter Classes 475
10.4.5. Actions 477
10.4.6. Mouse Events 482
10.4.7. The AWT Event Hierarchy 486
10.5. The Preferences API 490
11. User Interface Components with Swing 497
11.1. Swing and the Model-View-Controller Design Pattern 497
11.2. Introduction to Layout Management 501
11.2.1. Layout Managers 501
11.2.2. Border Layout 504
11.2.3. Grid Layout 506
11.3. Text Input 507
11.3.1. Text Fields 507
11.3.2. Labels and Labeling Components 509
11.3.3. Password Fields 510
11.3.4. Text Areas 511
11.3.5. Scroll Panes 512
11.4. Choice Components 513
11.4.1. Checkboxes 514
11.4.2. Radio Buttons 516
11.4.3. Borders 519
11.4.4. Combo Boxes 522
11.4.5. Sliders 525
11.5. Menus 530
11.5.1. Menu Building 530
11.5.2. Icons in Menu Items 533
11.5.3. Checkbox and Radio Button Menu Items 534
11.5.4. Pop-Up Menus 535
11.5.5. Keyboard Mnemonics and Accelerators 536
11.5.6. Enabling and Disabling Menu Items 539
11.5.7. Toolbars 543
11.5.8. Tooltips 545
11.6. The Grid Bag Layout 546
11.6.1. Grid Bag Basics 547
11.6.2. The `gridx`, `gridy`, `gridwidth`, and `gridheight` Parameters 548
11.6.3. Weight Fields 548
11.6.4. The `fill` and `anchor` Parameters 549
11.6.5. Padding 549
11.6.6. Alternative Method to Specify the `gridx`, `gridy`, `gridwidth`, and `gridheight` Parameters 549
11.6.7. A Grid Bag Layout Recipe 550
11.6.8. A Helper Class to Tame the Grid Bag Constraints 550
11.7. Custom Layout Managers 555
11.8. Dialog Boxes 558
11.8.1. Option Panes 559
11.8.2. Creating Dialogs 563
11.8.3. Data Exchange 566
11.8.4. File Dialogs 572

12. Advanced Swing and Graphics 581
12.1. Tables 581
12.1.1. A Simple Table 581
12.1.2. Table Models 585
12.2. Working with Rows and Columns 588
12.2.1. Column Classes 588
12.2.2. Accessing Table Columns 590
12.2.3. Resizing Columns 590
12.2.4. Resizing Rows 591
12.2.5. Selecting Rows, Columns, and Cells 592
12.2.6. Sorting Rows 592
12.2.7. Filtering Rows 594
12.2.8. Hiding and Displaying Columns 596
12.3. Cell Rendering and Editing 602
12.3.1. Rendering Cells 602
12.3.2. Rendering the Header 604
12.3.3. Editing Cells 604
12.3.4. Custom Editors 606
12.4. Trees 613
12.4.1. Simple Trees 614
12.4.2. Editing Trees and Tree Paths 623
12.4.3. Node Enumeration 629
12.4.4. Rendering Nodes 631
12.4.5. Listening to Tree Events 633
12.4.6. Custom Tree Models 639
12.5. Advanced AWT 647
12.5.1. The Rendering Pipeline 647
12.5.2. The `Shape` Class Hierarchy 650
12.5.3. Constructing Shapes 651
12.5.4. Areas 663
12.5.5. Strokes 664
12.5.6. Paint 671
12.5.7. Coordinate Transformations 673
12.5.8. Clipping 678
12.5.9. Transparency and Composition 680
12.6. Raster Images 688
12.6.1. Readers and Writers for Images 688
12.6.2. Obtaining Readers and Writers for Image File Types 688
12.6.3. Reading and Writing Files with Multiple Images 690
12.6.4. Image Manipulation 696
12.6.5. Filtering Images 703
12.7. Printing 710
12.7.1. Graphics Printing 710
12.7.2. Multiple-Page Printing 718
12.7.3. Print Services 726
12.7.4. Stream Print Services 729
12.7.5. Printing Attributes 731
13. The Foreign Functions and Memory API 741
13.1. Using JNI to Call C Code from a Java Program 742
13.2. Using FFM to Call a Foreign Function 747
13.3. Arenas 748

13.4. Memory Segments 749
13.5. Memory Layout 751
13.6. Looking Up and Invoking Foreign Functions 753
13.7. Callbacks 759
13.8. Advanced Topics 762
13.8.1. Accessing Memory With Variable Handles 762
13.8.2. Calling a Foreign Varargs Function 764
13.8.3. Capturing `errno` 764
Index *767*

Preface

To the Reader

The book you have in your hands is the second volume of the thirteenth edition of *Core Java*, fully updated for Java 25. 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.

With the explosive growth of the Java class library, a one-volume treatment of all the features of Java that serious programmers need to know is simply not possible. Hence, the book is broken up into two volumes.

This first volume concentrates on the fundamental concepts of the Java language:

- Object-oriented programming
- Reflection and proxies
- Interfaces and inner classes
- Exception handling
- Generic programming
- The collections framework
- Concurrency
- Annotations
- The Java platform module system

This second volume goes further into the most important libraries. For twelve editions, user interface programming was considered fundamental, but the time has come to recognize that it is no more, and to move it into the second volume. The volume includes detailed discussions of these topics:

- The Stream API
- File processing and regular expressions
- Databases
- XML processing
- Scripting and Compiling APIs
- Internationalization
- Network programming
- Graphical user interface design
- Graphics programming
- Foreign functions and memory

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 https://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.

A Tour of 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 executing computations in parallel with multiple cores.

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 the scripting and compiler APIs that allow your program to call code in scripting languages such as JavaScript or Groovy, and to compile Java code.

Chapter 9 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 10 provides an introduction into GUI programming. I show how you can make windows, how to paint on them, how to draw with geometric shapes, how to format text in multiple fonts, and how to display images. Next, you'll see how to write code that responds to events, such as mouse clicks or key presses.

Chapter 11 discusses the Swing GUI toolkit in great detail. The Swing toolkit allows you to build cross-platform graphical user interfaces. You'll learn all about the various kinds of buttons, text components, borders, sliders, list boxes, menus, and dialog boxes.

Chapter 12 contains advanced Swing material, 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 13 takes up foreign functions, which let you call code written in other programming languages. 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. Or you may want to call a battle-tested library for machine learning or data analysis.

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 25 are covered in detail.

Conventions

As is common in many computer books, I 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, I warn you with a "caution" icon.



Preview: Preview features that are slated to become a part of the language or API in the future are labeled with this icon.

Java comes with a large programming library, or Application Programming Interface (API). When using an API call for the first time, I add a short summary description at the end of the section. These descriptions are a bit more informal but, hopefully, 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, as shown in the following example:

java.lang.IO 25

- `println(Object obj)`
  Converts the object to a string and prints it on the console, followed by a line separator.

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

Listing 1.1: `NotHelloWorld.java`

```
1 void main() {
2     System.out.println("We will not use 'Hello, World!'");
3 }
```

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

Acknowledgments

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

My thanks go to my editor, Harry Misthos, and to Julie Nahil from Pearson for steering the book through the production process. I wrote the book using HTML and CSS, and Prince (https://princexml.com) turned it into PDF—a workflow that I highly recommend.

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), Alan Bateman (Oracle), Alec Beaton (IBM), Cliff Berg, Andrew Binstock (Oracle), Joshua Bloch, David Brown, Brian Burkhalter (Oracle), Corky Cartwright, Hillmer Chona, Frank Cohen (PushToTest), Chris Crane (devXsolution), Joe Darcy (Oracle), Nicholas J. De Lillo (Manhattan College), Rakesh Dhoopar (Oracle), Ahmad R. Elkomey, Hanno Embregts (Info Support), 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), Andrzej Grzesik, Marty Hall, Majid Hameed, Vincent Hardy (Adobe Systems), Dan Harkey (San Jose State University), Steve Haines, William Higgins (IBM), Marc Hoffmann (mtrail), Vladimir Ivanovic (PointBase), Jerry Jackson (CA Technologies), Josh Juneau, Heinz Kabutz (The Java Specialists' Newsletter, https://javaspecialists.eu), Stepan V. Kalinin (I-Teco/Servionica LTD), Tim Kimmet (Walmart), John Kostaras, Jerzy Krolak, Chris Laffra, Charlie Lai (Apple), Angelika Langer, Jeff Langr (Langr Software Solutions), Doug Langston, Hang Lau (McGill University), Mark Lawrence, Doug Lea (SUNY Oswego), Jason Lee (IBM), Gregory Longshore, Bob Lynch (Lynch Associates), Michael McMahon (Oracle), Rustam Mehmandarov, Philip Milne (consultant), Mark Morrissey (The Oregon Graduate Institute), Maurice Naftalin, 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), Naoto Sato (Oracle), 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, Henri Tremblay, Paul Tyma (consultant), Christian Ullenboom, Peter van der Linden, Joe Wang (Oracle), Sven Woltmann, Burt Walsh, Dan Xu (Oracle), and John Zavgren (Oracle).

Finally, a warm thank you to my coauthor of earlier editions, Gary Cornell, and to Greg Doench who was my editor for almost thirty years.

Cay Horstmann
Düsseldorf, Germany
September 2025

Chapter 1

Streams

Compared to collections, streams provide a view of data that lets you specify computations at a higher conceptual level. With a stream, you specify what you want to have done, not how to do it. You leave the scheduling of operations to the implementation. For example, suppose you want to compute the average of a certain property. You specify the source of data and the property, and the stream library can then optimize the computation, for example by using multiple threads for computing sums and counts and combining the results.

In this chapter, you will learn how to use the Java stream library, which allows you to process sequences of values in a “what, not how” style.

1.1. From Iterating to Stream Operations

When you process a collection, you usually iterate over its elements and do some work with each of them. For example, suppose we want to count all long words in a book. First, let's put them into a list:

```
String contents =  Files.readString(Path.of("alice.txt")); // Read file into string
String[] words = contents.split("\\PL+"); // Split into words
```

The `split` method splits a string into parts, given a regular expression for the delimiters. (See Chapter 2 for more information about regular expressions. For now, just take it on faith that the `\PL+` expression works.)

Now we are ready to iterate:

```
int count = 0;
for (String w : words) {
    if (w.length() > 12) count++;
}
```

With streams, the same operation looks like this:

```
long count = Stream.of(words)
    .filter(w -> w.length() > 12)
    .count();
```

Now you don't have to scan the loop for evidence of filtering and counting. The method names tell you right away what the code intends to do. Moreover, where the loop prescribes the order of operations in complete detail, a stream is able to schedule the operations any way it wants, as long as the result is correct.

Simply changing `stream` to `parallelStream` allows the stream library to do the filtering and counting in parallel.

```
long count = words.parallelStream()
    .filter(w -> w.length() > 12)
    .count();
```

Streams follow the "what, not how" principle. In our stream example, we describe what needs to be done: get the long words and count them. We don't specify in which order, or in which thread, this should happen. In contrast, the loop at the beginning of this section specifies exactly how the computation should work, and thereby forgoes any chances of optimization.

A stream seems superficially similar to a collection, allowing you to transform and retrieve data. But there are significant differences:

1. A stream does not store its elements. They may be stored in an underlying collection or generated on demand.
2. Stream operations don't mutate their source. For example, the `filter` method does not remove elements from a stream but yields a new stream in which they are not present.
3. Stream operations are *lazy* when possible. This means they are not executed until their result is needed. For example, if you only ask for the first five long words instead of all, the `filter` method will stop filtering after the fifth match. As a consequence, you can even have infinite streams!

Let us have another look at the example. The `stream` and `parallelStream` methods yield a *stream* for the `words` list. The `filter` method returns another stream that contains only the words of length greater than 12. The `count` method reduces that stream to a result.

This workflow is typical when you work with streams. You set up a pipeline of operations in three stages:

1. Create a stream.
2. Specify *intermediate operations* for transforming the initial stream into others, possibly in multiple steps.
3. Apply a *terminal operation* to produce a result. This operation forces the execution of the lazy operations that precede it. Afterwards, the stream can no longer be used.

In the example in Listing 1.1, the stream is created with the `stream` or `parallelStream` methods. The `filter` method transforms it, and `count` is the terminal operation.

In the next section, you will see how to create a stream. Three subsequent sections deal with intermediate operations. Then we turn to terminal operations.

Listing 1.1: v2ch01/streams/CountLongWords.java

```
1  package v2ch01.streams;
2
3  import module java.base;
4
5  class CountLongWords {
6      void main() throws Exception {
7          String contents = Files.readString(Path.of("gutenberg", "alice30.txt"));
8          String[] words = contents.split("\\PL+");
9
10         long count = 0;
11         for (String w : words) {
12             if (w.length() > 12) count++;
13         }
14         IO.println(count);
15
16         count = Stream.of(words).filter(w -> w.length() > 12).count();
17         IO.println(count);
18
19         count = Stream.of(words).parallel().filter(w -> w.length() > 12).count();
20         IO.println(count);
21     }
22 }
```

***java.util.stream.Stream<T>* 8**

- `Stream<T> filter(Predicate<? super T> p)`
  yields a stream containing all elements of this stream fulfilling `p`.
- `long count()`
  yields the number of elements of this stream. This is a terminal operation.

***java.util.Collection<E>* 1.2**

- `default Stream<E> stream()`
- `default Stream<E> parallelStream()`
  yield a sequential or parallel stream of the elements in this collection.

1.2. Stream Creation

You have already seen that you can turn any collection into a stream with the `stream` method of the `Collection` interface. If you have an array, use the static `Stream.of` method instead.

```
Stream<String> words = Stream.of(contents.split("\\PL+"));
```

In the preceding example, we make a stream from the array that the `split` method returns. The `of` method has a varargs parameter, so you can construct a stream from any number of arguments:

```
Stream<String> song = Stream.of("gently", "down", "the", "stream");
```

Use `Arrays.stream(array, from, to)` to make a stream from a part of an array.

To make a stream with no elements, use the static `Stream.empty` method:

```
Stream<String> silence = Stream.empty();
    // Generic type <String> is inferred; same as Stream.<String>empty()
```

The `Stream` interface has two static methods for making infinite streams. The `generate` method takes a function with no parameters (or, technically, an object of the `Supplier<T>` interface). Whenever a stream value is needed, that function is called to produce a value. You can get a stream of constant values as

```
Stream<String> echos = Stream.generate(() -> "Echo");
```

or a stream of random numbers as

```
Stream<Double> randoms = Stream.generate(Math::random);
```

To produce sequences such as `0 1 2 3 . . .`, use the `iterate` method instead. It takes a "seed" value and a function (technically, a `UnaryOperator<T>`) and repeatedly applies the function to the previous result. For example,

```
Stream<BigInteger> integers
    = Stream.iterate(BigInteger.ZERO, n -> n.add(BigInteger.ONE));
```

The first element in the sequence is the seed `BigInteger.ZERO`. The second element is `f(seed)` which yields 1 (as a big integer). The next element is `f(f(seed))` which yields 2, and so on.

To produce a finite stream instead, add a predicate that specifies when the iteration should finish:

```
var limit = new BigInteger("10000000");
Stream<BigInteger> integers
    = Stream.iterate(BigInteger.ZERO,
        n -> n.compareTo(limit) < 0,
        n -> n.add(BigInteger.ONE));
```

As soon as the predicate rejects an iteratively generated value, the stream ends.

Finally, the `Stream.ofNullable` method makes a really short stream from an object. The stream has length 0 if the object is `null` or length 1 otherwise, containing just the object. This is mostly useful in conjunction with `flatMap`—see Section 1.7.7 for an example.

A number of methods in the Java API yield streams. For example, the `String` class has a `lines` method that yields a stream of the lines contained in the string:

```
Stream<String> greetings = "Hello\nGuten Tag\nBonjour".lines();
```

The `Pattern` class has a method `splitAsStream` that splits a `CharSequence` by a regular expression. You can use the following statement to split a string into words:

```
Stream<String> words = Pattern.compile("\\PL+").splitAsStream(contents);
```

The `Scanner.tokens` method yields a stream of tokens of a scanner. Another way to get a stream of words from a string is

```
Stream<String> words = new Scanner(contents).tokens();
```

The static `Files.lines` method returns a `Stream` of all lines in a file:

```
try (Stream<String> lines = Files.lines(path)) {
    Process lines
}
```

Note that the try-with-resources block is necessary to close the file.

It is becoming more common to offer data as streams in the Java API. For example, Java 21 added a static `availableLocales` method that yields a stream of `Locale` objects.



Note: To produce a finite stream, you can always add the elements to a list which you then turn into a stream. The `Stream.Builder` interface is slightly more efficient for that purpose:

```
Stream<Integer> digits(int n) {
    Stream.Builder<Integer> builder = Stream.builder();
    while (n != 0) {
        builder.add(n % 10);
        n = n / 10;
    }
    return builder.build();
}
```

To view the contents of one of the streams introduced in this section, use the `toList` method, which collects the stream's elements in a list. Like `count`, `toList` is a terminal operation. If the stream is infinite, first truncate it with the `limit` method:

```
IO.println(Stream.generate(Math::random).limit(10).toList());
```



Note: In order to turn an `Iterator`, or a general `Iterable` that is not a collection, into a stream, you need to use helper methods for *spliterators*. A spliterator is similar to an iterator, but it can be split so that each resulting spliterator yields a part of the elements. This splitting is used for processing elements in parallel—see Section 1.13.

If you have an `Iterator` and want a stream of its results, use

```
StreamSupport.stream(Spliterators.spliteratorUnknownSize(
    iterator, Spliterator.ORDERED), false);
```

If you have an `Iterable` that is not a collection, you can turn it into a stream by calling

```
StreamSupport.stream(iterable.spliterator(), false);
```



Caution: It is very important that you don't modify the collection backing a stream while carrying out a stream operation. Remember that streams don't collect their data—the data is always in a separate collection. If you modify that collection, the outcome of the stream operations becomes undefined. The JDK documentation refers to this requirement as *noninterference*.

To be exact, since intermediate stream operations are lazy, it is possible to mutate the collection up to the point where the terminal operation executes. For example, the following, while certainly not recommended, will work:

```
List<String> wordList = . . .;
Stream<String> words = wordList.stream();
wordList.add("END");
long n = words.distinct().count();
```

But this code is wrong:

```
Stream<String> words = wordList.stream();
words.forEach(s -> {if (s.length() < 12) wordList.remove(s);});
    // ERROR--interference
```

The example program in Listing 1.2 shows the various ways of creating a stream.

Listing 1.2: `v2ch01/streams/CreatingStreams.java`

```
1  package v2ch01.streams;
2
3  import module java.base;
4
5  class CreatingStreams {
6      void main() throws Exception {
7          Path path = Path.of("gutenberg", "alice30.txt");
8          String contents = Files.readString(path);
9          Stream<String> words = Stream.of(contents.split("\\PL+"));
10
11         show("words", words);
12         Stream<String> song = Stream.of("gently", "down", "the", "stream");
13         show("song", song);
14         Stream<String> silence = Stream.empty();
15         show("silence", silence);
16
17         Stream<String> echos = Stream.generate(() -> "Echo");
18         show("echos", echos);
19
20         Stream<Double> randoms = Stream.generate(Math::random);
21         show("randoms", randoms);
22
23         Stream<BigInteger> integers = Stream.iterate(BigInteger.ONE,
24                 n -> n.add(BigInteger.ONE));
25         show("integers", integers);
26
```

```
27          Stream<String> greetings = "Hello\nGuten Tag\nBonjour".lines();
28          show("greetings", greetings);
29
30          Stream<String> wordsAnotherWay = Pattern.compile("\\PL+").splitAsStream(contents);
31          show("wordsAnotherWay", wordsAnotherWay);
32
33          try (Stream<String> lines = Files.lines(path)) {
34              show("lines", lines);
35          }
36
37          Iterable<Path> iterable = FileSystems.getDefault().getRootDirectories();
38          Stream<Path> rootDirectories = StreamSupport.stream(iterable.spliterator(), false);
39          show("rootDirectories", rootDirectories);
40
41          Iterator<Path> iterator = Path.of("/usr/share/dict/words").iterator();
42          Stream<Path> pathComponents = StreamSupport.stream(
43                  Spliterators.spliteratorUnknownSize(iterator, Spliterator.ORDERED), false);
44          show("pathComponents", pathComponents);
45      }
46
47      <T> void show(String title, Stream<T> stream) {
48          final int SIZE = 10;
49          List<T> firstElements = stream.limit(SIZE + 1).toList();
50          IO.print(title + ": ");
51          for (int i = 0; i < firstElements.size(); i++) {
52              if (i > 0) IO.print(", ");
53              if (i < SIZE)
54                  IO.print(firstElements.get(i));
55              else
56                  IO.print("...");
57          }
58          IO.println();
59      }
60
61  }
```

***java.util.stream.Stream<T>* 8**

- `static <T> Stream<T> of(T... values)`
  yields a stream whose elements are the given values.
- `static <T> Stream<T> empty()`
  yields a stream with no elements.
- `static <T> Stream<T> generate(Supplier<T> s)`
  yields an infinite stream whose elements are constructed by repeatedly invoking the function `s`.
- `static <T> Stream<T> iterate(T seed, UnaryOperator<T> f)`
- `static <T> Stream<T> iterate(T seed, Predicate<? super T> hasNext, UnaryOperator<T> f)`
  yield a stream whose elements are `seed`, `f` invoked on `seed`, `f` invoked on the preceding element, and so on. The first method yields an infinite stream. The stream of the second method comes to an end before the first element that doesn't fulfill the `hasNext` predicate.
- `static <T> Stream<T> ofNullable(T t)` **9**
  returns an empty stream if `t` is `null` or a stream containing `t` otherwise.

java.util.Spliterators 8

- `static <T> Spliterator<T> spliteratorUnknownSize(Iterator<? extends T> iterator, int characteristics)`
  turns an iterator into a splittable iterator of unknown size with the given characteristics (a bit pattern containing constants such as `Spliterator.ORDERED`).

java.util.Arrays 1.2

- `static <T> Stream<T> stream(T[] array, int startInclusive, int endExclusive)` **8**
  yields a stream whose elements are the specified range of the array.

java.lang.String

- `String[] split(String regex)` **1.4**
  finds all matches of the regular expression and splits the string by removing them. Returns an array with the fragments, but not a trailing empty string.
- `static Stream<String> lines()` **11**
  yields a stream whose elements are the lines of this string.

java.util.regex.Pattern 1.4

- `Stream<String> splitAsStream(CharSequence input)` **8**
  yields a stream whose elements are the parts of the input that are delimited by this pattern.

java.nio.file.Files 7

- `static Stream<String> lines(Path path)` **8**
- `static Stream<String> lines(Path path, Charset cs)` **8**
  yield a stream whose elements are the lines of the specified file, with the UTF-8 charset or the given charset.

java.util.stream.StreamSupport 8

- `static <T> Stream<T> stream(Spliterator<T> spliterator, boolean parallel)`
  yields a stream containing the values produced by the given splittable iterator.

***java.lang.Iterable* 5.0**

- `Spliterator<T> spliterator()` **8**
  yields a splittable iterator for this `Iterable`. The default implementation does not split and does not report a size.

java.util.Scanner 5.0

- `public Stream<String> tokens()` 9
  yields a stream of strings returned by calling the `next` method of this scanner.

java.util.function.Supplier<T> 8

- `T get()`
  supplies a value.

1.3. The `filter`, `map`, and `flatMap` Methods

A stream transformation produces a stream whose elements are derived from those of another stream. You have already seen the `filter` transformation that yields a new stream with those elements that match a certain condition. Here, we transform a stream of strings into another stream containing only long words:

```
List<String> wordList = . . .;
Stream<String> longWords = wordList.stream().filter(w -> w.length() > 12);
```

The parameter type of `filter` is `Predicate<T>`—that is, a function from `T` to `boolean`.

Often, you want to transform the values in a stream in some way. Use the `map` method and pass the function that carries out the transformation. For example, you can transform all words to lowercase like this:

```
Stream<String> lowercaseWords = words.stream().map(String::toLowerCase);
```

Here, we used `map` with a method reference. Often, you will use a lambda expression instead:

```
Stream<Character> firstCodeUnits = words.stream().map(s -> s.charAt(0));
```

The resulting stream contains the first code unit of each word.

When you use `map`, a function is applied to each element, yielding a new stream of the returned values. Now consider the situation where the returned values are themselves streams. The following method yields a stream of all grapheme clusters of a string.

```
public static Stream<String> graphemeClusters(String s) {
    return Stream.of(s.split("\\b{g}"));
}
```

For example, `graphemeClusters("Ahoy 🏴‍☠️")` is a stream of strings `"A"`, `"h"`, `"o"`, `"y"`, `" "`, and `"🏴‍☠️"`. (Note that the flag consists of multiple `char` values.)

Now let's map the `graphemeClusters` method on a stream of strings:

```
List<String> wordList = List.of(. . ., "your", "boat", . . .);
Stream<Stream<String>> result = wordList.stream().map(w -> graphemeClusters(w));
```

You will get a stream of streams, like `[. . . ["y", "o", "u", "r"], ["b", "o", "a", "t"], . . .]`. To flatten it out to a single stream `[. . . "y", "o", "u", "r", "b", "o", "a", "t", . . .]`, use the `flatMap` method instead of `map`:

```
Stream<String> flatResult = words.stream().flatMap(w -> graphemeClusters(w));
    // Calls graphemeClusters on each word and flattens the results
```

Sometimes, it is inefficient to produce a stream for each result sequence. The `mapMulti` method offers an alternative. Instead of producing a stream of results, you generate the results and pass them to a collector—an object of a class implementing the functional interface `Consumer`. For each result, invoke the collector's `accept` method.

Let's do this with an example. The following loop iterates over the grapheme clusters of a string `s`:

```
BreakIterator iter = BreakIterator.getCharacterInstance();
. . .
iter.setText(s);
int start = iter.first();
int end = iter.next();
while (end != BreakIterator.DONE) {
    String gc = s.substring(start, end);
    start = end;
    end = iter.next();
    // Do something with gc
}
```

When calling `mapMulti`, you provide a function that is invoked with the stream element and the collector. In your function, pass your results to the collector.

```
Stream<String> result = words.stream().mapMulti((s, collector) -> {
    iter.setText(s);
    int start = iter.first();
    int end = iter.next();
    while (end != BreakIterator.DONE) {
        String gc = s.substring(start, end);
        start = end;
        end = iter.next();
        collector.accept(gc);
    }
});
```

***java.util.stream.Stream<T>* 8**

- `Stream<T> filter(Predicate<? super T> predicate)`
  yields a stream containing the elements of this stream that fulfill the predicate.
- `<R> Stream<R> map(Function<? super T,? extends R> mapper)`
  yields a stream containing the results of applying `mapper` to the elements of this stream.

- `<R> Stream<R> flatMap(Function<? super T,? extends Stream<? extends R>> mapper)`
  yields a stream obtained by concatenating the results of applying `mapper` to the elements of this stream. (Note that each `mapper` result is a stream.)
- `<R> Stream<R> mapMulti(BiConsumer<? super T,? super Consumer<R>> mapper)` **16**
  For each stream element, the `mapper` is invoked, and all elements passed to the `Consumer` during invocation are added to the result stream.

1.4. Extracting Substreams and Combining Streams

The call *stream*`.limit(n)` returns a new stream that ends after `n` elements (or when the original stream ends if it is shorter). This method is particularly useful for cutting infinite streams down to size. For example,

```
Stream<Double> randoms = Stream.generate(Math::random).limit(100);
```

yields a stream with 100 random numbers.

The call *stream*`.skip(n)` does the exact opposite. It discards the first `n` elements. This is handy in our book reading example where, due to the way the `split` method works, the first element is an unwanted empty string. We can make it go away by calling `skip`:

```
Stream<String> words = Stream.of(contents.split("\\PL+")).skip(1);
```

The *stream*`.takeWhile(`*predicate*`)` call takes all elements from the stream while the predicate is `true`, and then stops.

For example, suppose we use the `graphemeClusters` method of the preceding section to split a string into characters, and we want to collect all initial digits. The `takeWhile` method can do this:

```
Stream<String> initialDigits = graphemeClusters(str).takeWhile(
    s -> "0123456789".contains(s));
```

The `dropWhile` method does the opposite, dropping elements while a condition is `true` and yielding a stream of all elements starting with the first one for which the condition was `false`. For example,

```
Stream<String> withoutInitialWhiteSpace = graphemeClusters(str).dropWhile(
    s -> s.strip().length() == 0);
```

You can concatenate two streams with the static `concat` method of the `Stream` class:

```
Stream<String> combined = Stream.concat(
    graphemeClusters("Hello"), graphemeClusters("World"));
    // Yields the stream ["H", "e", "l", "l", "o", "W", "o", "r", "l", "d"]
```

Of course, the first stream should not be infinite—otherwise the second one wouldn't ever get a chance.

***java.util.stream.Stream<T>* 8**

- `Stream<T> limit(long maxSize)`
  yields a stream with up to `maxSize` of the initial elements from this stream.
- `Stream<T> skip(long n)`
  yields a stream whose elements are all but the initial `n` elements of this stream.
- `Stream<T> takeWhile(Predicate<? super T> predicate)` **9**
  yields a stream whose elements are the initial elements of this stream that fulfill the predicate.
- `Stream<T> dropWhile(Predicate<? super T> predicate)` **9**
  yields a stream whose elements are the elements of this stream except for the initial ones that fulfill the predicate.
- `static <T> Stream<T> concat(Stream<? extends T> a, Stream<? extends T> b)`
  yields a stream whose elements are the elements of `a` followed by the elements of `b`.

1.5. Other Stream Transformations

The `distinct` method returns a stream that yields elements from the original stream, in the same order, except that duplicates are suppressed. The duplicates need not be adjacent.

```
Stream<String> uniqueWords
    = Stream.of("merrily", "merrily", "merrily", "gently").distinct();
    // Only one "merrily" is retained
```

For sorting a stream, there are several variations of the `sorted` method. One works for streams of `Comparable` elements, and another accepts a `Comparator`. Here, we sort strings so that the longest ones come first:

```
Stream<String> longestFirst
    = words.stream().sorted(Comparator.comparing(String::length).reversed());
```

As with all stream transformations, the `sorted` method yields a new stream whose elements are the elements of the original stream in sorted order.

Of course, you can sort a collection without using streams. The `sorted` method is useful when the sorting process is part of a stream pipeline.

Finally, the `peek` method yields another stream with the same elements as the original, but a function is invoked every time an element is retrieved. That is handy for debugging:

```
Object[] powers = Stream.iterate(1.0, p -> p * 2)
    .peek(e -> IO.println("Fetching " + e))
    .limit(20)
    .toArray();
```

When an element is actually accessed, a message is printed. This way you can verify that the infinite stream returned by `iterate` is processed lazily.



Tip: When you use a debugger to debug a stream computation, you can set a breakpoint in a method that is called from one of the transformations. With most IDEs, you can also set breakpoints in lambda expressions. If you just want to know what happens at a particular point in the stream pipeline, add

```
.peek(x -> {
    return;
})
```

and set a breakpoint on the second line.

***java.util.stream.Stream<T>* 8**

- `Stream<T> distinct()`
  yields a stream of the distinct elements of this stream.
- `Stream<T> sorted()`
- `Stream<T> sorted(Comparator<? super T> comparator)`
  yield a stream whose elements are the elements of this stream in sorted order. The first method requires that the elements are instances of a class implementing `Comparable`.
- `Stream<T> peek(Consumer<? super T> action)`
  yields a stream with the same elements as this stream, passing each element to `action` as it is consumed.

1.6. Simple Reductions

Now that you have seen how to create and transform streams, we will finally get to the most important point—getting answers from the stream data. The methods covered in this section are called *reductions*. Reductions are *terminal operations*. They reduce the stream to a nonstream value that can be used in your program.

You have already seen a simple reduction: the `count` method that returns the number of elements of a stream.

Other simple reductions are `max` and `min` that return the largest or smallest value. There is a twist—these methods return an `Optional<T>` value that either wraps the answer or indicates that there is none (because the stream happened to be empty). In the olden days, it was common to return `null` in such a situation. But that can lead to null pointer exceptions when it happens in an incompletely tested program. The `Optional` type is a better way of indicating a missing return value. We discuss the `Optional` type in detail in the next section. Here is how you can get the maximum of a stream:

```
Optional<String> largest = words.max(String::compareToIgnoreCase);
IO.println("largest: " + largest.orElse(""));
```

The `findFirst` returns the first value in a nonempty collection. It is often useful when combined with `filter`. For example, here we find the first word that starts with the letter Q, if it exists:

```
Optional<String> startsWithQ
    = words.filter(s -> s.startsWith("Q")).findFirst();
```

If you are OK with any match, not just the first one, use the `findAny` method. This is effective when you parallelize the stream, since the stream can report any match that it finds instead of being constrained to the first one.

```
Optional<String> startsWithQ
    = words.parallel().filter(s -> s.startsWith("Q")).findAny();
```

If you just want to know if there is a match, use the terminal `anyMatch` operation with a predicate argument:

```
boolean aWordStartsWithQ
    = words.parallel().anyMatch(s -> s.startsWith("Q"));
```

There are methods `allMatch` and `noneMatch` that return `true` if all or no elements match a predicate. These methods also benefit from being run in parallel.

***java.util.stream.Stream<T>* 8**

- `Optional<T> max(Comparator<? super T> comparator)`
- `Optional<T> min(Comparator<? super T> comparator)`
  yield a maximum or minimum element of this stream, using the ordering defined by the given comparator, or an empty `Optional` if this stream is empty. These are terminal operations.
- `Optional<T> findFirst()`
- `Optional<T> findAny()`
  yield the first, or any, element of this stream, or an empty `Optional` if this stream is empty. These are terminal operations.
- `boolean anyMatch(Predicate<? super T> predicate)`
- `boolean allMatch(Predicate<? super T> predicate)`
- `boolean noneMatch(Predicate<? super T> predicate)`
  return `true` if any, all, or none of the elements of this stream match the given predicate. These are terminal operations.

1.7. The Optional Type

An `Optional<T>` object is a wrapper for either an object of type `T` or no object. In the former case, we say that the value is *present*. The `Optional<T>` type is intended as a safer alternative for a reference of type `T` that either refers to an object or is `null`. But it is only safer if you use it right. The next three sections show you how.

1.7.1. Getting an Optional Value

The key to using `Optional` effectively is to use a method that either *produces an alternative* if the value is not present, or *consumes the value* only if it is present.

In this section, we look at the first strategy. Often, there is a default that you want to use when there is no match, perhaps the empty string:

```
String result = optionalString.orElse("");
    // The wrapped string, or "" if none
```

You can also invoke code to compute the default:

```
String result = optionalString.orElseGet(() -> System.getProperty("myapp.default"));
    // The function is only called when needed
```

Or you can throw an exception if there is no value:

```
String result = optionalString.orElseThrow(IllegalStateException::new);
    // Supply a method that yields an exception object
```

java.util.Optional 8

- `T orElse(T other)`
  yields the value of this `Optional`, or `other` if this `Optional` is empty.
- `T orElseGet(Supplier<? extends T> other)`
  yields the value of this `Optional`, or the result of invoking `other` if this `Optional` is empty.
- `<X extends Throwable> T orElseThrow(Supplier<? extends X> exceptionSupplier)`
  yields the value of this `Optional`, or throws the result of invoking `exceptionSupplier` if this `Optional` is empty.

1.7.2. Consuming an Optional Value

In the preceding section, you saw how to produce an alternative if no value is present. The other strategy for working with optional values is to consume the value only if it is present.

The `ifPresent` method accepts a function. If the optional value exists, it is passed to that function. Otherwise, nothing happens.

```
optionalValue.ifPresent(v -> Process v);
```

For example, if you want to add the value to a set if it is present, call

```
optionalValue.ifPresent(v -> results.add(v));
```

or simply

```
optionalValue.ifPresent(results::add);
```

If you want to take one action if the `Optional` has a value and another action if it doesn't, use `ifPresentOrElse`:

```
optionalValue.ifPresentOrElse(
    v -> IO.println("Found " + v),
    () -> logger.warning("No match"));
```

java.util.Optional 8

- `void ifPresent(Consumer<? super T> action)`
  If this `Optional` is nonempty, passes its value to `action`.
- `void ifPresentOrElse(Consumer<? super T> action, Runnable emptyAction) 9`
  If this `Optional` is nonempty, passes its value to `action`, else invokes `emptyAction`.

1.7.3. Pipelining Optional Values

In the preceding sections, you saw how to get a value out of an `Optional` object. Another useful strategy is to keep the `Optional` intact. You can transform the value inside an `Optional` by using the `map` method:

```
Optional<Path> transformed = optionalString.map(Path::of);
```

If `optionalString` is empty, then `transformed` is also empty.



Note: This `map` method is the analog of the `map` method of the `Stream` interface that you have seen in Section 1.3. Simply imagine an optional value as a stream of size zero or one. The result again has size zero or one, and in the latter case, the function has been applied.

Similarly, you can use the `filter` method to only consider `Optional` values that fulfill a certain property before or after transforming it. If the property is not fulfilled, the pipeline yields an empty result:

```
Optional<Path> transformed = optionalString
    .filter(s -> s.endsWith(".txt"))
    .map(Path::of);
```

You can substitute an alternative `Optional` for an empty `Optional` with the `or` method. The alternative is computed lazily.

```
Optional<String> result = optionalString.or(() -> // Supply an Optional
    alternatives.stream().findFirst());
```

If `optionalString` has a value, then `result` is `optionalString`. If not, the lambda expression is evaluated, and its result is used.

java.util.Optional 8

- `<U> Optional<U> map(Function<? super T,? extends U> mapper)`
  yields an `Optional` whose value is obtained by applying the given function to the value of this `Optional` if present, or an empty `Optional` otherwise.
- `Optional<T> filter(Predicate<? super T> predicate)`
  yields an `Optional` with the value of this `Optional` if it fulfills the given predicate, or an empty `Optional` otherwise.

- `Optional<T> or(Supplier<? extends Optional<? extends T>> supplier)` **9**
  yields this `Optional` if it is nonempty, or the one produced by the supplier otherwise.

1.7.4. How Not to Work with Optional Values

If you don't use `Optional` values correctly, you have no benefit over the "something or `null`" approach of the past.

The `get` method gets the wrapped element of an `Optional` value if it exists, or throws a `NoSuchElementException` if it doesn't. Therefore,

```
Optional<T> optionalValue = . . .;
optionalValue.get().someMethod()
```

is no safer than

```
T value = . . .;
value.someMethod();
```

The `isPresent` and `isEmpty` methods report whether or not an `Optional<T>` object has a value. But

```
if (optionalValue.isPresent()) optionalValue.get().someMethod();
```

is no easier than

```
if (value != null) value.someMethod();
```



Tip: If you find yourself using the `get`, `isPresent`, or `isEmpty` methods, try to rethink your approach and use one of the strategies of the preceding two sections.



Note: The `orElseThrow` method is a scarier-sounding synonym for the `get` method. When you call `optionalValue.orElseThrow().someMethod()`, it becomes explicit that an exception will occur if the `optionalValue` is empty. The hope is that programmers will only use `orElseThrow` when it is absolutely clear that this cannot happen.

Here are a few more tips for the proper use of the `Optional` type:

- A variable of type `Optional` should *never* be `null`.
- Don't use fields of type `Optional`. The cost is an additional object. Inside a class, use `null` for an absent field. To discourage `Optional` fields, the class is not serializable.
- Method parameters of type `Optional` are questionable. They make the call unpleasant in the common case where the requested value is present. Instead, consider two overloaded versions of the method, with and without the parameter. (On the other hand, returning an `Optional` is fine. It is the proper way to indicate that a function may not have a result.)
- Don't put `Optional` objects in a set, and don't use them as keys for a map. Collect the values instead.

java.util.Optional 8

- `T get()`
- `T orElseThrow()` **10**
  yield the value of this `Optional`, or throw a `NoSuchElementException` if it is empty.
- `boolean isEmpty()` **11**
- `boolean isPresent()`
  return `true` if this `Optional` is empty or not empty.

1.7.5. Creating Optional Values

So far, we have discussed how to consume an `Optional` object someone else created. If you want to write a method that creates an `Optional` object, there are several static methods for that purpose, including `Optional.of(result)` and `Optional.empty()`. For example,

```
public static Optional<Double> inverse(Double x) {
    return x == 0 ? Optional.empty() : Optional.of(1 / x);
}
```

The `ofNullable` method is intended as a bridge from possibly `null` values to optional values. `Optional.ofNullable(obj)` returns `Optional.of(obj)` if `obj` is not `null` and `Optional.empty()` otherwise.

java.util.Optional 8

- `static <T> Optional<T> of(T value)`
- `static <T> Optional<T> ofNullable(T value)`
  yield an `Optional` with the given value. If `value` is `null`, the first method throws a `NullPointerException` and the second method yields an empty `Optional`.
- `static <T> Optional<T> empty()`
  yields an empty `Optional`.

1.7.6. Composing Optional Value Functions with `flatMap`

Suppose you have a method `f` yielding an `Optional<T>`, and the target type `T` has a method `g` yielding an `Optional<U>`. If they were normal methods, you could compose them by calling `s.f().g()`. But that composition doesn't work since `s.f()` has type `Optional<T>`, not `T`. Instead, call

```
Optional<U> result = s.f().flatMap(T::g);
```

If `s.f()` is present, then `g` is applied to it. Otherwise, an empty `Optional<U>` is returned.

Clearly, you can repeat that process if you have more methods or lambdas that yield `Optional` values. You can then build a pipeline of steps, simply by chaining calls to `flatMap`, that will succeed only when all parts do.

For example, consider the safe `inverse` method of the preceding section. Suppose we also have a safe square root:

```
public static Optional<Double> squareRoot(Double x) {
    return x < 0 ? Optional.empty() : Optional.of(Math.sqrt(x));
}
```

Then you can compute the square root of the `inverse` as

```
Optional<Double> result = inverse(arg).flatMap(x -> squareRoot(x));
```

We can make such a chain look a little more regular by starting out with an `Optional`:

```
Optional<Double> result
    = Optional.of(arg).flatMap(x -> inverse(x)).flatMap(x -> squareRoot(x));
```

It looks even neater with method expressions:

```
Optional<Double> result
    = Optional.of(arg).flatMap(this::inverse).flatMap(this::squareRoot);
```

If either the `inverse` method or the `squareRoot` returns `Optional.empty()`, the result is empty.



Note: In this example, taken from Listing 1.3, the `inverse` and `squareRoot` methods are instance methods. Hence the method expressions start with `this::` . Had these been static methods, they would be prefixed by *ClassName*`::`.



Note: You have already seen a `flatMap` method in the `Stream` interface (see Section 1.3). That method maps a stream-yielding method to all elements of a stream, and then flattens out the resulting stream of streams. The `Optional.flatMap` method works in the same way. By mapping an optional-yielding method to an optional, you get an optional of optional, which is then flattened out.

`java.util.Optional` 8

- `<U> Optional<U> flatMap(Function<? super T,? extends Optional<? extends U>> mapper)` yields the result of applying `mapper` to the value in this `Optional` if present, or an empty optional otherwise.

1.7.7. Turning an Optional into a Stream

The `stream` method turns an `Optional<T>` into a `Stream<T>` with zero or one element. Sure, why not, but why would you ever want that?

This becomes useful with methods that return an `Optional` result. Suppose you have a stream of user IDs and a method

```
Optional<User> lookup(String id)
```

How do you get a stream of users, skipping those IDs that are invalid?

Of course, you can filter out the invalid IDs and then apply `get` to the remaining ones:

```
Stream<String> ids = . . .;
Stream<User> users = ids.map(Users::lookup)
    .filter(Optional::isPresent)
    .map(Optional::get);
```

But that uses the `isPresent` and `get` methods that we warned about. It is more elegant to call

```
Stream<User> users = ids.map(Users::lookup)
    .flatMap(Optional::stream);
```

Each call to `stream` returns a stream with zero or one element. The `flatMap` method combines them all. That means the nonexistent users are simply dropped.



Note: In this section, we consider the happy scenario in which we have a method that returns an `Optional` value. These days, many methods return `null` when there is no valid result. Suppose `Users.classicLookup(id)` returns a `User` object or `null`, not an `Optional<User>`. Then you can of course filter out the `null` values:

```
Stream<User> users = ids.map(Users::classicLookup)
    .filter(Objects::nonNull);
```

But if you prefer the `flatMap` approach, you can use

```
Stream<User> users = ids.flatMap(
    id -> Stream.ofNullable(Users.classicLookup(id)));
```

or

```
Stream<User> users = ids.map(Users::classicLookup)
    .flatMap(Stream::ofNullable);
```

The call `Stream.ofNullable(obj)` yields an empty stream if `obj` is `null` or a stream just containing `obj` otherwise.

The example program in Listing 1.3 demonstrates the `Optional` API.

Listing 1.3: v2ch01/optional/OptionalDemo.java

```
1  package v2ch01.optional;
2
3  import module java.base;
4
5  class OptionalDemo {
6      void main() throws Exception {
7          String contents = Files.readString(Path.of("gutenberg", "alice30.txt"));
8          List<String> wordList = List.of(contents.split("\\PL+"));
9
10         Optional<String> optionalValue = wordList
```

```
                .stream()
                .filter(s -> s.contains("fred"))
                .findFirst();
        IO.println(optionalValue.orElse("No word") + " contains fred");

        Optional<String> optionalString = Optional.empty();
        String result = optionalString.orElse("N/A");
        IO.println("result: " + result);
        result = optionalString.orElseGet(() -> Locale.getDefault().getDisplayName());
        IO.println("result: " + result);
        try {
            result = optionalString.orElseThrow(IllegalStateException::new);
            IO.println("result: " + result);
        }
        catch (Throwable t) {
            t.printStackTrace(System.out);
        }

        optionalValue = wordList.stream().filter(s -> s.contains("red")).findFirst();
        optionalValue.ifPresent(s -> IO.println(s + " contains red"));

        var results = new HashSet<String>();
        optionalValue.ifPresent(results::add);
        Optional<Boolean> added = optionalValue.map(results::add);
        IO.println(added);

        IO.println(inverse(4.0).flatMap(this::squareRoot));
        IO.println(inverse(-1.0).flatMap(this::squareRoot));
        IO.println(inverse(0.0).flatMap(this::squareRoot));
        Optional<Double> result2 = Optional.of(-4.0)
                .flatMap(this::inverse)
                .flatMap(this::squareRoot);
        IO.println(result2);
    }

    Optional<Double> inverse(Double x) {
        return x == 0 ? Optional.empty() : Optional.of(1 / x);
    }

    Optional<Double> squareRoot(Double x) {
        return x < 0 ? Optional.empty() : Optional.of(Math.sqrt(x));
    }
}
```

java.util.Optional 8

- `<U> Optional<U> flatMap(Function<? super T,Optional<U>> mapper)` 9
  yields the result of applying `mapper` to the value of this `Optional`, or an empty `Optional` if this `Optional` is empty.

1.8. Collecting Results

When you are done with a stream, you will often want to look at the results. You can call the `iterator` method, which yields an old-fashioned iterator that you can use to visit the elements.

Alternatively, you can call the `forEach` method to apply a function to each element:

```
stream.forEach(IO::println);
```

On a parallel stream, the `forEach` method traverses elements in arbitrary order. If you want to process them in stream order, call `forEachOrdered` instead. Of course, you might then give up some or all of the benefits of parallelism.

But more often than not, you will want to collect the result in a data structure. You have already seen the `toList` method that yields a list of the stream elements.

Call `toArray` to get an array of the stream elements.

Since it is not possible to create a generic array at runtime, the expression `stream.toArray()` returns an `Object[]` array. If you want an array of the correct type, pass in the array constructor:

```
String[] result = stream.toArray(String[]::new);
    // stream.toArray() has type Object[]
```

To collect results into other data structures, the `Stream` interface has a generic `collect` method. You can concatenate the results in a string, store them in any collection, and insert them into maps. By implementing your own collector, you can consume the stream data in arbitrary ways. You will see the details in the following sections.

***java.util.stream.BaseStream* 8**

- `Iterator<T> iterator()`
  yields an iterator for obtaining the elements of this stream. This is a terminal operation.

***java.util.stream.Stream<T>* 8**

- `List<T> toList()` **16**
  yields a list of the elements in this stream.
- `void forEach(Consumer<? super T> action)`
  invokes `action` on each element of the stream.
- `Object[] toArray()`
- `<A> A[] toArray(IntFunction<A[]> generator)`
  yield an array of objects, or of type `A` when passed a constructor reference `A[]::new`. These are terminal operations.
- `<R,A> R collect(Collector<? super T,A,R> collector)`
  collects the elements in this stream, using the given collector. The `Collectors` class has factory methods for many collectors.

1.9. Collectors

For collecting stream elements to another target, there is a convenient `collect` method that takes an instance of the `Collector` interface. A *collector* is an object that accumulates elements and produces a result. The `Collectors` class provides a large number of factory methods for common collectors.

Suppose you want to collect all strings in a stream by concatenating them. You can call

```
String result = stream.collect(Collectors.joining());
```

If you want a delimiter between elements, pass it to the `joining` method:

```
String result = stream.collect(Collectors.joining(", "));
```

If your stream contains objects other than strings, you need to first convert them to strings, like this:

```
String result = stream.map(Object::toString).collect(Collectors.joining(", "));
```

If you want to reduce the stream results to a sum, count, average, maximum, or minimum, use one of the `summarizing(Int|Long|Double)` methods. These methods take a function that maps the stream objects to numbers and yield a result of type `(Int|Long|Double)SummaryStatistics`, simultaneously computing the sum, count, average, maximum, and minimum.

```
IntSummaryStatistics summary = stream.collect(
    Collectors.summarizingInt(String::length));
double averageWordLength = summary.getAverage();
double maxWordLength = summary.getMax();
```

Note that you are not calling the `joining` and `summarizingInt` methods on a stream. These are static methods of the `Collectors` class. They yield instances of `Collector` which you pass to the `collect` method of the `Stream` interface.

The `joining` collector yields a string, and the summarizing collectors yield four numbers. However, most collectors collect values in data structures, as you will see in the following sections.

The example program in Listing 1.5 shows how to collect elements from a stream.

Listing 1.4: `v2ch01/collecting/CollectingResults.java`

```
1  package v2ch01.collecting;
2
3  import module java.base;
4
5  class CollectingResults {
6      void main() throws Exception {
7          // Turning a stream into an iterator
8          Iterator<Integer> iter = Stream.iterate(0, n -> n + 1).limit(10).iterator();
9          while (iter.hasNext())
10             IO.println(iter.next());
11
12         Object[] numbers = Stream.iterate(0, n -> n + 1).limit(10).toArray();
13         IO.println("Object array: " + numbers + " " + Arrays.toString(numbers));
14         // Note it's an Object[] array
15
16         try {
17             var number = (Integer) numbers[0]; // OK
18             IO.println("number: " + number);
19             IO.println("The following statement throws an exception:");
```

```
            var _ = (Integer[]) numbers; // Throws exception
        }
        catch (ClassCastException e) {
            IO.println(e);
        }

        Integer[] numbers3 = Stream.iterate(0, n -> n + 1).limit(10).toArray(Integer[]::new);
        IO.println("Integer array: " + numbers3 + " " + Arrays.toString(numbers3));
        // Note it's an Integer[] array

        String result = noVowels().limit(10).collect(Collectors.joining());
        IO.println("Joining: " + result);
        result = noVowels().limit(10).collect(Collectors.joining(", "));
        IO.println("Joining with commas: " + result);

        IntSummaryStatistics summary = noVowels()
                .collect(Collectors.summarizingInt(String::length));
        double averageWordLength = summary.getAverage();
        double maxWordLength = summary.getMax();
        IO.println("Average word length: " + averageWordLength);
        IO.println("Max word length: " + maxWordLength);
        IO.println("forEach:");
        noVowels().limit(10).forEach(IO::println);
    }

    Stream<String> noVowels() throws IOException {
        String contents = Files.readString(Path.of("gutenberg", "alice30.txt"));
        Stream<String> words = Stream.of(contents.split("\\PL+"));
        return words.map(s -> s.replaceAll("[aeiouAEIOU]", ""));
    }

    <T> void show(String label, Set<T> set) {
        IO.print(label + ": " + set.getClass().getName());
        IO.println("["
            + set.stream().limit(10).map(Object::toString).collect(Collectors.joining(", "))
            + "]");
    }
}
```

java.util.stream.Collectors 8

- `static Collector<CharSequence,?,String> joining()`
- `static Collector<CharSequence,?,String> joining(CharSequence delimiter)`
- `static Collector<CharSequence,?,String> joining(CharSequence delimiter, CharSequence prefix, CharSequence suffix)`
  yield a collector that joins strings. The delimiter is placed between strings, and the prefix and suffix before the first and after the last string. When not specified, these are empty.
- `static <T> Collector<T,?,IntSummaryStatistics> summarizingInt(ToIntFunction<? super T> mapper)`
- `static <T> Collector<T,?,LongSummaryStatistics> summarizingLong(ToLongFunction<? super T> mapper)`
- `static <T> Collector<T,?,DoubleSummaryStatistics> summarizingDouble(ToDoubleFunction<? super T> mapper)`
  yield collectors that produce an `(Int|Long|Double)SummaryStatistics` object, from which you can obtain the count, sum, average, maximum, and minimum of the results of applying `mapper` to each element.

java.util.IntSummaryStatistics 8

java.util.LongSummaryStatistics 8

java.util.DoubleSummaryStatistics 8

- `long getCount()`
  yields the count of the summarized elements.
- *xxx* `getSum()`
- `double getAverage()`
  yield the sum or average of the summarized elements, or zero if there are no elements.
- *xxx* `getMax()`
- *xxx* `getMin()`
  yield the maximum or minimum of the summarized elements, or `(Integer|Long|Double).(MAX|MIN)_VALUE` if there are no elements.

1.9.1. Collecting into Collections

Before the `toList` method was added in Java 16, you had to use the collector produced by `Collectors.toList()`:

```
List<String> result = stream.collect(Collectors.toList());
```

Similarly, here is how you can collect stream elements into a set:

```
Set<String> result = stream.collect(Collectors.toSet());
```

These calls give you a list or set, but you cannot make any further assumptions. The collection might not be mutable, serializable, or thread-safe. If you want to control which kind of collection you get, use the following call instead:

```
TreeSet<String> result = stream.collect(Collectors.toCollection(TreeSet::new));
```

The example program in Listing 1.5 shows how to place stream elements into a collection.

Listing 1.5: v2ch01/collecting/CollectingResults.java

```
1  package v2ch01.collecting;
2
3  import module java.base;
4
5  class CollectingResults {
6      void main() throws Exception {
7          // Turning a stream into an iterator
8          Iterator<Integer> iter = Stream.iterate(0, n -> n + 1).limit(10).iterator();
9          while (iter.hasNext())
10             IO.println(iter.next());
11
12         Object[] numbers = Stream.iterate(0, n -> n + 1).limit(10).toArray();
13         IO.println("Object array: " + numbers + " " + Arrays.toString(numbers));
```

```
        // Note it's an Object[] array

        try {
            var number = (Integer) numbers[0]; // OK
            IO.println("number: " + number);
            IO.println("The following statement throws an exception:");
            var _ = (Integer[]) numbers; // Throws exception
        }
        catch (ClassCastException e) {
            IO.println(e);
        }

        Integer[] numbers3 = Stream.iterate(0, n -> n + 1).limit(10).toArray(Integer[]::new);
        IO.println("Integer array: " + numbers3 + " " + Arrays.toString(numbers3));
        // Note it's an Integer[] array

        String result = noVowels().limit(10).collect(Collectors.joining());
        IO.println("Joining: " + result);
        result = noVowels().limit(10).collect(Collectors.joining(", "));
        IO.println("Joining with commas: " + result);

        IntSummaryStatistics summary = noVowels()
                .collect(Collectors.summarizingInt(String::length));
        double averageWordLength = summary.getAverage();
        double maxWordLength = summary.getMax();
        IO.println("Average word length: " + averageWordLength);
        IO.println("Max word length: " + maxWordLength);
        IO.println("forEach:");
        noVowels().limit(10).forEach(IO::println);
    }

    Stream<String> noVowels() throws IOException {
        String contents = Files.readString(Path.of("gutenberg", "alice30.txt"));
        Stream<String> words = Stream.of(contents.split("\\PL+"));
        return words.map(s -> s.replaceAll("[aeiouAEIOU]", ""));
    }

    <T> void show(String label, Set<T> set) {
        IO.print(label + ": " + set.getClass().getName());
        IO.println("["
            + set.stream().limit(10).map(Object::toString).collect(Collectors.joining(", "))
            + "]");
    }
}
```

java.util.stream.Collectors 8

- static <T> Collector<T,?,List<T>> toList()
- static <T> Collector<T,?,List<T>> toUnmodifiableList() 10
- static <T> Collector<T,?,Set<T>> toSet()
- static <T> Collector<T,?,Set<T>> toUnmodifiableSet() 10

yield collectors that collect elements in a list or set.

- `static <T,C extends Collection<T>> Collector<T,?,C> toCollection(Supplier<C> collectionFactory)`
  yields a collector that collects elements into an arbitrary collection. Pass a constructor reference such as `TreeSet::new`.

1.9.2. Collecting into Maps

Suppose you have a `Stream<Person>` and want to collect the elements into a map so that later you can look up people by their ID. Call `Collectors.toMap` with two functions that produce the map's keys and values. For example,

```
public record Person(int id, String name) {}
. . .
Map<Integer, String> idToName = people.collect(
    Collectors.toMap(Person::id, Person::name));
```

In the common case when the values should be the actual elements, use `Function.identity()` for the second function.

```
Map<Integer, Person> idToPerson = people.collect(
    Collectors.toMap(Person::id, Function.identity()));
```

If there is more than one element with the same key, there is a conflict, and the collector will throw an `IllegalStateException`. You can override that behavior by supplying a third function that resolves the conflict and determines the value for the key, given the existing and the new value. Your function could return the existing value, the new value, or a combination of them.

Here, we construct a map that contains, for each language in the available locales, as key its name in your default locale (such as `"German"`), and as value its localized name (such as `"Deutsch"`).

```
Map<String, String> languageNames = Locale.availableLocales().collect(
    Collectors.toMap(
        Locale::getDisplayLanguage,
        loc -> loc.getDisplayLanguage(loc),
        (existingValue, newValue) -> existingValue));
```

We don't care that the same language might occur twice (for example, German in Germany and in Switzerland), so we just keep the first entry.



Note: In this chapter, I use the `Locale` class as a source of an interesting data set. See Chapter 7 for more information on working with locales.

Now suppose we want to know all languages in a given country. Then we need a `Map<String, Set<String>>`. For example, the value for `"Switzerland"` is the set `[French, German, Italian]`. At first, we store a singleton set for each language. Whenever a new language is found for a given country, we form the union of the existing and the new set.

```
Map<String, Set<String>> countryLanguageSets = Locale.availableLocales().collect(
    Collectors.toMap(
        Locale::getDisplayCountry,
        l -> Collections.singleton(l.getDisplayLanguage()),
        (a, b) -> { // Union of a and b
            var union = new HashSet<String>(a);
            union.addAll(b);
            return union;
        }));
```

You will see a simpler way of obtaining this map in the next section.

If you want a `TreeMap`, supply the constructor as the fourth argument. You must provide a merge function. Here is one of the examples from the beginning of the section, now yielding a `TreeMap`:

```
Map<Integer, Person> idToPerson = people.collect(
    Collectors.toMap(
        Person::id,
        Function.identity(),
        (existingValue, newValue) -> { throw new IllegalStateException(); },
        TreeMap::new));
```



Note: For each of the `toMap` methods, there is an equivalent `toConcurrentMap` method that yields a concurrent map. A single concurrent map is used in the parallel collection process. When used with a parallel stream, a shared map is more efficient than merging maps. Note that elements are no longer collected in stream order, but that doesn't usually make a difference.

The program in Listing 1.6 gives examples of collecting stream results into maps.

Listing 1.6: `v2ch01/collecting/CollectingIntoMaps.java`

```
1  package v2ch01.collecting;
2
3  import module java.base;
4
5  record Person(int id, String name) {}
6
7  class CollectingIntoMaps {
8      void main() throws Exception {
9          Map<Integer, String> idToName = people()
10                 .collect(Collectors.toMap(Person::id, Person::name));
11         IO.println("idToName: " + idToName);
12
13         Map<Integer, Person> idToPerson = people()
14                 .collect(Collectors.toMap(Person::id, Function.identity()));
15         IO.println("idToPerson: " + idToPerson.getClass().getName() + idToPerson);
16
17         idToPerson = people().collect(
18                 Collectors.toMap(Person::id, Function.identity(), (_, _) -> {
19                     throw new IllegalStateException();
```

```
20                  }, TreeMap::new));
21          IO.println("idToPerson: " + idToPerson.getClass().getName() + idToPerson);
22
23          Map<String, String> languageNames = Locale.availableLocales()
24                  .collect(Collectors.toMap(Locale::getDisplayLanguage,
25                          l -> l.getDisplayLanguage(l),
26                          (existingValue, _) -> existingValue));
27          IO.println("languageNames: " + languageNames);
28
29          Map<String, Set<String>> countryLanguageSets = Locale.availableLocales()
30                  .collect(Collectors.toMap(Locale::getDisplayCountry,
31                          l -> Set.of(l.getDisplayLanguage()), (a, b) -> { // union of a and b
32                              Set<String> union = new HashSet<>(a);
33                              union.addAll(b);
34                              return union;
35                          }));
36          IO.println("countryLanguageSets: " + countryLanguageSets);
37      }
38
39      static Stream<Person> people() {
40          return Stream.of(new Person(1001, "Peter"), new Person(1002, "Paul"),
41                  new Person(1003, "Mary"));
42      }
43  }
```

java.util.stream.Collectors 8

- static <T,K,U> Collector<T,?,Map<K,U>> toMap(Function<? super T,? extends K> keyMapper, Function<? super T,? extends U> valueMapper)
- static <T,K,U> Collector<T,?,Map<K,U>> toMap(Function<? super T,? extends K> keyMapper, Function<? super T,? extends U> valueMapper, BinaryOperator<U> mergeFunction)
- static <T,K,U,M extends Map<K,U>> Collector<T,?,M> toMap(Function<? super T,? extends K> keyMapper, Function<? super T,? extends U> valueMapper, BinaryOperator<U> mergeFunction, Supplier<M> mapSupplier)
- static <T,K,U> Collector<T,?,Map<K,U>> toUnmodifiableMap(Function<? super T,? extends K> keyMapper, Function<? super T,? extends U> valueMapper) **10**
- static <T,K,U> Collector<T,?,Map<K,U>> toUnmodifiableMap(Function<? super T,? extends K> keyMapper, Function<? super T,? extends U> valueMapper, BinaryOperator<U> mergeFunction) **10**
- static <T,K,U> Collector<T,?,ConcurrentMap<K,U>> toConcurrentMap(Function<? super T,? extends K> keyMapper, Function<? super T,? extends U> valueMapper)
- static <T,K,U> Collector<T,?,ConcurrentMap<K,U>> toConcurrentMap(Function<? super T,? extends K> keyMapper, Function<? super T,? extends U> valueMapper, BinaryOperator<U> mergeFunction)
- static <T,K,U,M extends ConcurrentMap<K,U>> Collector<T,?,M> toConcurrentMap(Function<? super T,? extends K> keyMapper, Function<? super T,? extends U> valueMapper, BinaryOperator<U> mergeFunction, Supplier<M> mapSupplier)

  yield a collector that produces a map, unmodifiable map, or concurrent map. The `keyMapper` and `valueMapper` functions are applied to each collected element, yielding a key/value entry of the resulting map. By default, an `IllegalStateException` is thrown when two elements give rise to the same key. You can instead supply a `mergeFunction` that merges values with the same key. By default, the result is a `HashMap` or `ConcurrentHashMap`. You can instead supply a `mapSupplier` that yields the desired map instance.

1.9.3. Grouping and Partitioning

In the preceding section, you saw how to collect all languages in a given country. But the process was a bit tedious. You had to generate a singleton set for each map value and then specify how to merge the existing and new values. Forming groups of values with the same characteristic is very common, so the `groupingBy` method supports it directly.

Let's look at the problem of grouping locales by country. First, form this map:

```
Map<String, List<Locale>> countryToLocales = Locale.availableLocales().collect(
    Collectors.groupingBy(Locale::getCountry));
```

The function `Locale::getCountry` is the *classifier function* of the grouping. You can now look up all locales for a given country code, for example

```
List<Locale> swissLocales = countryToLocales.get("CH");
    // Yields locales de_CH, fr_CH, it_CH, and maybe more
```



Note: A quick refresher on locales: Each locale has a language code (such as `en` for English) and a country code (such as `US` for the United States). The locale `en_US` describes English in the United States, and `en_IE` is English in Ireland. Some countries have multiple locales. For example, `ga_IE` is Gaelic in Ireland, and, as the preceding example shows, the JDK knows at least three locales in Switzerland.

When the classifier function is a predicate function (that is, a function returning a `boolean` value), the stream elements are partitioned into two lists: those where the function returns `true` and the complement. In this case, it is more efficient to use `partitioningBy` instead of `groupingBy`. For example, here we split all locales into those that use English and all others:

```
Map<Boolean, List<Locale>> englishAndOtherLocales = Locale.availableLocales().collect(
    Collectors.partitioningBy(l -> l.getLanguage().equals("en")));
List<Locale> englishLocales = englishAndOtherLocales.get(true);
```



Note: If you call the `groupingByConcurrent` method, you get a concurrent map that, when used with a parallel stream, is concurrently populated. This is entirely analogous to the `toConcurrentMap` method.

`java.util.stream.Collectors` 8

- `static <T,K> Collector<T,?,Map<K,List<T>>> groupingBy(Function<? super T,? extends K> classifier)`
- `static <T,K> Collector<T,?,ConcurrentMap<K,List<T>>> groupingByConcurrent(Function<? super T,? extends K> classifier)`

  yield a collector that produces a map or concurrent map whose keys are the results of applying `classifier` to all collected elements, and whose values are lists of elements with the same key.

- `static <T> Collector<T,?,Map<Boolean,List<T>>> partitioningBy(Predicate<? super T> predicate)` yields a collector that produces a map whose keys are `true`/`false`, and whose values are lists of the elements that fulfill/do not fulfill the predicate.

1.9.4. Downstream Collectors

The `groupingBy` method yields a map whose values are lists. If you want to process those lists in some way, supply a *downstream collector*. For example, if you want sets instead of lists, you can use the `Collectors.toSet` collector that you saw in the preceding section:

```
Map<String, Set<Locale>> countryToLocaleSet = Locale.availableLocales().collect(
    groupingBy(Locale::getCountry, toSet()));
```



Note: In this example, as well as the remaining examples of this section, I assume a static import of `java.util.stream.Collectors.*` to make the expressions easier to read.

You can also apply `groupingBy` twice:

```
Map<String, Map<String, List<Locale>>> countryAndLanguageToLocale =
    Locale.availableLocales().collect(
        groupingBy(Locale::getCountry,
            groupingBy(Locale::getLanguage)));
```

Then `countryAndLanguageToLocale.get("IN").get("hi")` is a list of the Hindi locales in India. (There are several variants.)

Several collectors are provided for reducing collected elements to numbers:

- `counting` produces a count of the collected elements. For example,

  ```
  Map<String, Long> countryToLocaleCounts = Locale.availableLocales().collect(
      groupingBy(Locale::getCountry, counting()));
  ```

  counts how many locales there are for each country.
- `summing(Int|Long|Double)` and `averaging(Int|Long|Double)` apply a provided function to the downstream elements and produce the sum or average of the function's results. For example,

  ```
  public record City(String name, String state, int population) {}
  . . .
  Map<String, Integer> stateToCityPopulation = cities.collect(
      groupingBy(City::state, averagingInt(City::population)));
  ```

  computes the average of populations per state in a stream of cities.
- `maxBy` and `minBy` take a comparator and produce maximum and minimum of the downstream elements. For example,

```
Map<String, Optional<City>> stateToLargestCity = cities.collect(
    groupingBy(City::state,
        maxBy(Comparator.comparing(City::population))));
```

produces the largest city per state.

The `collectingAndThen` collector adds a final processing step behind a collector. For example, if you want to know how many distinct results there are, collect them into a set and then compute the size:

```
Map<Character, Integer> stringCountsByStartingLetter = strings.collect(
    groupingBy(s -> s.charAt(0),
        collectingAndThen(toSet(), Set::size)));
```

The `mapping` collector does the opposite. It applies a function to each collected element and passes the results to a downstream collector.

```
Map<Character, Set<Integer>> stringLengthsByStartingLetter = strings.collect(
    groupingBy(s -> s.charAt(0),
        mapping(String::length, toSet())));
```

Here, we group strings by their first character. Within each group, we produce the lengths and collect them in a set.

The `mapping` method also yields a nicer solution to a problem from the preceding section—gathering a set of all languages in a country.

```
Map<String, Set<String>> countryToLanguages = Locale.availableLocales().collect(
    groupingBy(Locale::getDisplayCountry,
        mapping(Locale::getDisplayLanguage,
            toSet())));
```

There is a `flatMapping` method as well, for use with functions that return streams.

If the grouping or mapping function has return type `int`, `long`, or `double`, you can collect elements into a summary statistics object, as discussed in Section 1.8. For example,

```
Map<String, IntSummaryStatistics> stateToCityPopulationSummary = cities.collect(
    groupingBy(City::state,
        summarizingInt(City::population)));
```

Then you can get the sum, count, average, minimum, and maximum of the function values from the summary statistics objects of each group.

The `filtering` collector applies a filter to each group, for example:

```
Map<String, Set<City>> largeCitiesByState
    = cities.collect(
        groupingBy(City::state,
            filtering(c -> c.population() > 500000,
                toSet()))); // States without large cities have empty sets
```

Finally, you can use the `teeing` collector to branch into two downstream collections. This is useful whenever you need to compute more than one result from a stream. Suppose you want to collect city names and also compute their average population. You can't read a stream twice, but `teeing` lets you carry out two computations. Specify two downstream collectors and a function that combines the results.

```
record Pair<S, T>(S first, T second) {}
Pair<List<String>, Double> result = cities.filter(c -> c.state().equals("NV"))
    .collect(teeing(
        mapping(City::name, toList()), // First downstream collector
        averagingDouble(City::population), // Second downstream collector
        (list, avg) -> new Pair(list,  avg))); // Combining function
```



Note: There are also three versions of a `reducing` method that apply general reductions, as described in the next section.

Composing collectors is powerful, but it can lead to very convoluted expressions. The best use is with `groupingBy` or `partitioningBy` to process the "downstream" map values. Otherwise, simply apply methods such as `map`, `reduce`, `count`, `max`, or `min` directly on streams.

The example program in Listing 1.7 demonstrates downstream collectors.

Listing 1.7: v2ch01/collecting/DownstreamCollectors.java

```
1  package v2ch01.collecting;
2
3  import static java.util.stream.Collectors.*;
4
5  import module java.base;
6
7  record City(String name, String state, int population) {}
8
9  class DownstreamCollectors {
10
11     void main() throws Exception {
12         Map<String, Set<Locale>> countryToLocaleSet = Locale.availableLocales()
13                 .collect(groupingBy(Locale::getCountry, toSet()));
14         IO.println("countryToLocaleSet: " + countryToLocaleSet);
15
16         Map<String, Map<String, List<Locale>>> countryAndLanguageToLocale = Locale
17                 .availableLocales()
18                 .collect(groupingBy(Locale::getCountry, groupingBy(Locale::getLanguage)));
19         IO.println(
20             "Hindi locales in India: " + countryAndLanguageToLocale.get("IN").get("hi"));
21
22         Map<String, Long> countryToLocaleCounts = Locale.availableLocales()
23                 .collect(groupingBy(Locale::getCountry, counting()));
24         IO.println("countryToLocaleCounts: " + countryToLocaleCounts);
25
26         Stream<City> cities = readCities("v2ch01/cities.txt");
27         Map<String, Integer> stateToCityPopulation = cities
```

```
                .collect(groupingBy(City::state, summingInt(City::population)));
        IO.println("stateToCityPopulation: " + stateToCityPopulation);

        cities = readCities("v2ch01/cities.txt");
        Map<String, Optional<String>> stateToLongestCityName = cities.collect(groupingBy(
            City::state, mapping(City::name, maxBy(Comparator.comparing(String::length)))));
        IO.println("stateToLongestCityName: " + stateToLongestCityName);

        Map<String, Set<String>> countryToLanguages = Locale.availableLocales()
                .collect(groupingBy(Locale::getDisplayCountry,
                    mapping(Locale::getDisplayLanguage, toSet())));
        IO.println("countryToLanguages: " + countryToLanguages);

        cities = readCities("v2ch01/cities.txt");
        Map<String, IntSummaryStatistics> stateToCityPopulationSummary = cities
                .collect(groupingBy(City::state, summarizingInt(City::population)));
        IO.println(stateToCityPopulationSummary.get("NY"));

        cities = readCities("v2ch01/cities.txt");
        Map<String, String> stateToCityNames = cities.collect(groupingBy(City::state,
            reducing("", City::name, (s, t) -> s.length() == 0 ? t : s + ", " + t)));

        cities = readCities("v2ch01/cities.txt");
        stateToCityNames = cities
                .collect(groupingBy(City::state, mapping(City::name, joining(", "))));
        IO.println("stateToCityNames: " + stateToCityNames);

        cities = readCities("v2ch01/cities.txt");
        record Pair<S, T>(S first, T second) {
        }
        Pair<List<String>, Double> result = cities.filter(c -> c.state().equals("NV")).collect(
            teeing(mapping(City::name, toList()), averagingDouble(City::population),
                (names, avg) -> new Pair<>(names, avg)));
        IO.println(result);
    }

    Stream<City> readCities(String filename) throws IOException {
        return Files.lines(Path.of(filename)).map(l -> l.split(", "))
                .map(a -> new City(a[0], a[1], Integer.parseInt(a[2])));
    }
}
```

java.util.stream.Collectors 8

- `public static <T,K,A,D> Collector<T,?,Map<K,D>> groupingBy(Function<? super T,? extends K> classifier, Collector<? super T,A,D> downstream)`
  yields a collector that produces a map. The keys are the results of applying `classifier` to all collected elements. The values are the results of collecting elements with the same key, using the downstream collector.
- `static <T> Collector<T,?,Long> counting()`
  yields a collector that counts the collected elements.

- `static <T> Collector<T,?,Integer> summingInt(ToIntFunction<? super T> mapper)`
- `static <T> Collector<T,?,Long> summingLong(ToLongFunction<? super T> mapper)`
- `static <T> Collector<T,?,Double> summingDouble(ToDoubleFunction<? super T> mapper)`
  yield a collector that computes the sum of the results of applying `mapper` to the collected elements.
- `static <T> Collector<T,?,Optional<T>> maxBy(Comparator<? super T> comparator)`
- `static <T> Collector<T,?,Optional<T>> minBy(Comparator<? super T> comparator)`
  yield a collector that computes the maximum or minimum of the collected elements, using the ordering specified by `comparator`.
- `static <T,A,R,RR> Collector<T,A,RR> collectingAndThen(Collector<T,A,R> downstream, Function<R,RR> finisher)`
  yields a collector that sends elements to the downstream collector and then applies the finisher function to its result.
- `static <T,U,A,R> Collector<T,?,R> mapping(Function<? super T,? extends U> mapper, Collector<? super U,A,R> downstream)`
  yields a collector that calls `mapper` on each element and passes the results to the downstream collector.
- `static <T,U,A,R> Collector<T,?,R> flatMapping(Function<? super T,? extends Stream<? extends U>> mapper, Collector<? super U,A,R> downstream)`
  yields a collector that calls `mapper` on each element and passes the elements of the results to the downstream collector.
- `static <T,A,R> Collector<T,?,R> filtering(Predicate<? super T> predicate, Collector<? super T,A,R> downstream)`
  yields a collector that passes the elements fulfilling the predicate to the downstream collector.

1.9.5. Implementing Collectors

In the collection process, a collector accumulates incoming stream elements in an internal data structure called a "result container." Each collector can choose a suitable result container type. If the stream is parallel, multiple result containers are filled concurrently and then merged. After accumulation and merging, the collector can optionally transform the final result container into another object that is the collection result.

Specifically, a `Collector` must implement four methods, each of which yields a function object:

- `Supplier<A> supplier()` supplies a result container of type `A`.
- `BiConsumer<A,T> accumulator()` accumulates a stream element of type `T` into a result container.
- `BinaryOperator<A> combiner()` combines two result containers into one.
- `Function<A, R> finisher()` transforms the final result container into the result of type `R`.

Let's look at the function objects of the `Collectors.toList()` collector:

- The supplier is a function yielding an empty `ArrayList<T>` or simply `ArrayList<T>::new`.
- The accumulator is `(rc, t) -> rc.add(t)` or `List::add`. (That method expression is why the result container is the first parameter.)
- The combiner concatenates two lists: `(rc1, rc2) -> { rc1.addAll(rc2); return rc1; }`.
- The `finisher` does nothing (which is the most common case).

The `characteristics` method of a collector returns a set of `Collector.Characteristics` flags that can help the stream with optimizing the collection process. There are three of them:

- `IDENTITY_FINISH`: the finisher need not be called.
- `UNORDERED`: it is ok not to call the accumulator in encounter order.
- `CONCURRENT`: it is safe to call the accumulator from multiple threads on the same result container.

`Collectors.joining` returns a collector with none of these characteristics. The result container is a `StringBuilder`. The finisher must convert it into a string. Accumulation order matters. And the result container is not threadsafe.

In contrast, the `toConcurrentMap` collector has all three characteristics. The result container is a threadsafe `ConcurrentHashMap`. Insertion order does not matter, and no finishing is needed.



Caution: Collectors are meant to work efficiently with parallel streams, which makes them unsuitable for certain terminal operations. Consider collecting every *n*th element from a stream. That can't be parallelized, since the accumulator can't track the indexes in each segment. But a collector can't tell the stream that it wants the elements sequentially.

In this situation, you are better off using the gatherer API—see Section 1.11.

Let's build a collector that statistically samples elements, accepting them with a given probability. For now, we'll collect the accepted elements in a list.

Following the `Collectors` API, let's make a method that yields the collector:

```
<T> Collector<T, List<T>, List<T>> randomSampling(double p) {
   return Collector.of(ArrayList<T>::new,
           (resultContainer, e) -> {
               if (Math.random() < p)
                   resultContainer.add(e);
               },
           (rc1, rc2) -> { rc1.addAll(rc2); return rc1; },
           Function.identity(),
           Collector.Characteristics.IDENTITY_FINISH);
   }
```

The `Collector.of` method has five parameters: the four functions and a varargs parameter for the characteristics.

This collector is almost the same as the `toList` collector, except that the accumulator only stores some of the elements.

Here is how to use it:

```
Stream<Integer> numbers = . . .;
List<Integer> sampled = numbers.collect(randomSampling(0.05)); // About every 20th element
```

What if we don't want the accepted elements in a list? Our API should support downstream collectors. Here is how:

```
<T, A, R> Collector<T, A, R> randomSampling(double p, Collector<T, A, R> downstream) {
        BiConsumer<A, T> downstreamAccumulator = downstream.accumulator();
        return Collector.of(downstream.supplier(),
                (resultContainer, e) -> {
                    if (Math.random() < p)
                        downstreamAccumulator.accept(resultContainer, e);
                    },
                downstream.combiner(),
                downstream.finisher(),
                downstream.characteristics().toArray(Collector.Characteristics[]::new));
    }
```

Here, `downstream` is a collector that accepts elements of type `T`, with a result container of type `A`, and result type `R`.

We can reuse the downstream's supplier, combiner, and finisher, but we must adapt its accumulator, so that only a fraction of the incoming elements are inserted.

Unfortunately, the API leaves something to be desired for this particular use case. I have to turn the downstream's set of characteristics into an array for the varargs parameter.

The program in Listing 1.8 has the complete example.

Listing 1.8: `v2ch01/collecting/ImplementingCollectors.java`

```
1  package v2ch01.collecting;
2
3  import module java.base;
4
5  class ImplementingCollectors {
6      void main() {
7          IO.println(IntStream.range(1, 1000)
8              .boxed()
9              .parallel()
10             .collect(randomSampling(0.05)));
11         IO.println(IntStream.range(1, 1000)
12             .boxed()
13             .parallel()
14             .collect(randomSampling(0.05, Collectors.toSet())));
15     }
16
17     <T> Collector<T, List<T>, List<T>> randomSampling(double p) {
18         return Collector.of(ArrayList<T>::new,
19                 (resultContainer, e) -> {
20                     if (Math.random() < p)
21                         resultContainer.add(e);
22                     },
23                 (rc1, rc2) -> { rc1.addAll(rc2); return rc1; },
24                 Function.identity(),
25                 Collector.Characteristics.IDENTITY_FINISH);
```

```
    }

    <T, A, R> Collector<T, A, R> randomSampling(double p, Collector<T, A, R> downstream) {
        BiConsumer<A, T> downstreamAccumulator = downstream.accumulator();
        return Collector.of(downstream.supplier(),
                (resultContainer, e) -> {
                    if (Math.random() < p)
                        downstreamAccumulator.accept(resultContainer, e);
                    },
                downstream.combiner(),
                downstream.finisher(),
                downstream.characteristics().toArray(Collector.Characteristics[]::new));
    }
}
```

java.util.stream.Collector<T, A, R> 8

- Supplier<A> supplier()
  The function to supply a new result container.
- BiConsumer<A,T> accumulator()
  The function to incorporate an element from the stream into a result container.
- BinaryOperator<A> combiner()
  The function to combine two result containers.
- Function<A,R> finisher()
  The function that turns the final result container into the result of the collector
- Set<Collector.Characteristics> characteristics()
  The characteristics of this collector.
- static <T,A,R> Collector<T,A,R> of(Supplier<A> supplier, BiConsumer<A,T> accumulator, BinaryOperator<A> combiner, Function<A,R> finisher, Collector.Characteristics... characteristics)
  yields a collector with the given supplier, accumulator, combiner, finisher, and characteristics.

1.10. Reduction Operations

The reduce method is another general mechanism for computing a value from a stream. In particular with mathematical operations, it is much easier to specify a reduction than it would be to implement the same operation with a collector. The simplest form takes a binary function and keeps applying it, starting with the first two elements. It's easy to explain this if the function is the sum:

```
List<Integer> values = . . .;
Optional<Integer> sum = values.stream().reduce((x, y) -> x + y);
```

In this case, the reduce method computes $v_0 + v_1 + v_2 + \ldots$, where v_i are the stream elements. The method returns an Optional because there is no valid result if the stream is empty.



Note: In this case, you can write reduce(Integer::sum) instead of reduce((x, y) -> x + y).

More generally, you can use any operation that combines a partial result `x` with the next value `y` to yield a new partial result.

Here is another way of looking at reductions. Given a reduction operation *op*, the reduction yields v_0 *op* v_1 *op* v_2 *op* ..., where v_i *op* v_{i+1} denotes the function call $op(v_i, v_{i+1})$. There are many operations that might be useful in practice—such as sum, product, string concatenation, maximum and minimum, set union or intersection.

If you want to use reduction with parallel streams, the operation must be *associative*: It shouldn't matter in which order you combine the elements. In math notation, (*x op y*) *op z* must be equal to *x op* (*y op z*). An example of an operation that is not associative is subtraction. For example, $(6 - 3) - 2 \neq 6 - (3 - 2)$.

Often, there is an *identity e* such that *e op x* = *x*, and that element can be used as the start of the computation. For example, 0 is the identity for addition, and you can use the second form of `reduce`:

```
List<Integer> values = . . .;
Integer sum = values.stream().reduce(0, (x, y) -> x + y);
    // Computes 0 + v0 + v1 + v2 + . . .
```

The identity value is returned if the stream is empty, and you no longer need to deal with the `Optional` class.

Now suppose you have a stream of objects and want to form the sum of some property, such as lengths in a stream of strings. You can't use the simple form of `reduce`. It requires a function `(T, T) -> T`, with the same types for the parameters and the result, but in this situation you have two types: The stream elements have type `String`, and the accumulated result is an integer. There is a form of `reduce` that can deal with this situation.

First, you supply an "accumulator" function `(total, word) -> total + word.length()`. That function is called repeatedly, forming the cumulative total. But when the computation is parallelized, there will be multiple computations of this kind, and you need to combine their results. You supply a second function for that purpose. The complete call is

```
int result = words.reduce(0,
    (total, word) -> total + word.length(),
    (total1, total2) -> total1 + total2);
```



Note: In practice, you probably won't use the `reduce` method a lot. It is usually easier to map to a stream of numbers and use one of its methods to compute sum, maximum, or minimum. (We discuss streams of numbers in Section 1.12.) In this particular example, you could have called `words.mapToInt(String::length).sum()`, which is both simpler and more efficient since it doesn't involve boxing.



Note: There are times when `reduce` is not general enough. For example, suppose you want to collect the results in a `BitSet`. If the collection is parallelized, you can't put the elements directly into a single `BitSet` because a `BitSet` object is not thread-safe. For that reason, you

can't use `reduce`. Each segment needs to start out with its own empty set, and `reduce` only lets you supply one identity value. Instead, use the three-parameter form of `collect`. You provide three functions:

1. A *supplier* to construct new instances of the target object
2. An *accumulator* that adds an element to the target
3. A *combiner* that merges two target objects into one

Here is how the `collect` method works for a bit set:

```
BitSet result = stream.collect(BitSet::new, BitSet::set, BitSet::or);
```

***java.util.Stream<T>* 8**

- `Optional<T> reduce(BinaryOperator<T> accumulator)`
- `T reduce(T identity, BinaryOperator<T> accumulator)`
- `<U> U reduce(U identity, BiFunction<U,? super T,U> accumulator, BinaryOperator<U> combiner)`
  form a cumulative total of the stream elements with the given `accumulator` function. If `identity` is provided, then it is the first value to be accumulated. If `combiner` is provided, it can be used to combine totals of segments that are accumulated separately.
- `<R> R collect(Supplier<R> supplier, BiConsumer<R,? super T> accumulator, BiConsumer<R,R> combiner)`
  collects elements in a result of type `R`. On each segment, `supplier` is called to provide an initial result, `accumulator` is called to mutably add elements to it, and `combiner` is called to combine two results.

1.11. Gatherers

The stream API has an extension point for implementing arbitrary *terminal* operations: the `collect` method, whose argument is a `Collector`. Many collector implementations are provided in the `Collectors` class. You can also implement your own, as you have seen in Section 1.9.5.

Java 24 adds an extension point for intermediate operations. The `gather` method, whose argument is a `Gatherer`, turns a stream into another stream.

1.11.1. Predefined Gatherers

The `Gatherers` class provides a few `Gatherer` implementations. The `windowFixed` and `windowSliding` gatherers group adjacent elements together, yielding a stream of lists. It's easier to show them in action than to explain them in words:

```
IntStream.range(0, 10).boxed().gather(Gatherers.windowFixed(4)).toList()
    // [[0, 1, 2, 3], [4, 5, 6, 7], [8, 9]]
IntStream.range(0, 10).boxed().gather(Gatherers.windowSliding(4)).toList()
    // [[0, 1, 2, 3], [1, 2, 3, 4], [2, 3, 4, 5], [3, 4, 5, 6],
    // [4, 5, 6, 7], [5, 6, 7, 8], [6, 7, 8, 9]]
```

The `windowSliding` gatherer is useful for comparing adjacent elements of a stream. For example, here is how to drop adjacent duplicates:

```
Stream.concat(words, Stream.of((String) null))
    .gather(Gatherers.windowSliding(2))
    .filter(w -> !w.get(0).equalsIgnoreCase(w.get(1)))
    .map(w -> w.get(0))
```

By appending a `null` to the stream, the last element can be compared with something.

The `mapConcurrent` gatherer is similar to `map`, but the computations run in *virtual* threads. This is very useful for blocking operations. For example, if you have a list of URLs, you can read them concurrently.

```
String read(String url) {
    try {
        return new String(URI.create(url).toURL().openStream().readAllBytes());
    }
    catch (IOException e) {
        throw new UncheckedIOException(e);
    }
}

int max = 100; // Maximumum number of concurrent threads
List<String> contents = urls.stream()
    .gather(Gatherers.mapConcurrent(max, this::read))
    .toList();
```

If one of the computations throws an exception, all remaining tasks are cancelled, and the exception is rethrown.



Note: As you will see in Section 1.13, parallel streams are used to parallelize computationally intensive operations over multiple processor cores. In contrast, the `mapConcurrent` gatherer is intended for concurrent operations that mostly block.

Finally, there are `fold` and `scan` gatherers. A *fold* is a general mechanism for repeatedly applying an operation *op*. It starts with an initial value *iv* and computes $(((iv\ op\ v_0)\ op\ v_1)\ op\ v_2\ op)\ \ldots$, where v_i are the stream elements.

This looks similar to a reduction, which you saw in Section 1.10. However, reductions are intended for parallelizable operations. A fold is inherently sequential, and the operation need not be associative.

Some functional programmers love folds because any loop can be translated into a fold. Consider this computation of a number from its decimal digits:

```
int n = 0;
for (int d : digits)
    n = 10 * n + d;
```

Here is the same computation as a fold:

```
int n = digits.stream()
    .gather(Gatherers.fold(() -> 0, (x, y) -> x * 10 + y))
    .findFirst()
    .orElse(0);
```

The `fold` gatherer produces a stream with one element, or none if `digits` was empty.

As you can see from the example, you have to provide a supplier for the initial value, and the operation that is repeatedly applied.



Note: It seems odd that folding is implemented as a gatherer and not a collector. It would be more natural as a terminal operation since it produces a single result. But collectors are unable to signal that they must receive stream elements sequentially. As you will see in Section 1.11.2, gatherers have this capability. But a gatherer must yield a stream; in this case, one of length zero or one.

There is also a `scan` gatherer that produces a stream of all intermediate results. For example,

```
Stream.of(1, 7, 2, 9)
    .gather(Gatherers.scan(() -> 0, (x, y) -> x * 10 + y))
    .toList() // [1, 17, 172, 1729]
```

1.11.2. Implementing Gatherers

Implementing a gatherer is similar to implementing a collector (Section 1.9.5). A gatherer has a generic "intermediate state" object instead of the "result collection" of a collector. It too has four methods yielding function objects. Here `T` is the type of the stream elements, `A` the type of the intermediate state, and `R` the element type of the stream that the gatherer produces.

- `Supplier<A> initializer()` supplies an intermediate state instance.
- `Gatherer.Integrator<A, T, R> integrator()` processes a stream element.
- `BinaryOperator<A> combiner()` combines two intermediate states int one.
- `BiConsumer<A, Gatherer.Downstream<? super R>> finisher()` carries out a final operation after processing all stream elements.

`Gatherer.Integrator` is a functional interface with this method:

```
boolean integrate(A state, T element, Gatherer.Downstream<? super R> downstream)
```

In the integrator and finisher, you call the `push` method of the `Gatherer.Downstream` interface in order to send values to the stream that the gatherer produces. You can call `push` as often as you like, or not at all.

Very simple gatherers can be stateless. Then the initializer is not needed.

A combiner is only needed for parallel streams.

The finisher is only needed if you need to push values after the last element has been integrated. For example, the `windowFixed` gatherer pushes the short list of remaining elements in the finisher.

As an example, let us implement a gatherer that samples every *n*th element of a stream.

Here is how we want to use it:

```
List<String> samples = wordList.stream().gather(sampling(50)).toList();
```

The result is a list holding every 50th word from the original list.

The `sampling` method yields the gatherer:

```
static class IntermediateState {
    long index = -1;
}

<T> Gatherer<T, IntermediateState, T> sampling(int n) {
    return Gatherer.ofSequential(
        IntermediateState::new,
        Gatherer.Integrator.of((state, e, downstream) -> {
            state.index++;
            if (state.index % n == 0)
                return downstream.push(e);
            else
                return true;
            }));
    }
```

The `Gatherer.ofSequential` method yields a gatherer that calls the integrator in encounter order. There is also a `Gatherer.of` method for constructing parallelizable gatherers.

This gatherer doesn't have a combiner or finisher. In general, you don’t provide arguments for unneeded initializers, combiners and finishers. Use one of the overloaded versions with just the gatherer operations you need.



Tip: Use these factory methods rather than declaring an explicit class implementing the `Gatherer` interface. It is less work, and it is easier to tell whether your gatherer is sequential or parallelizable.

You may wonder how the stream implementation distinguishes the two. A gatherer is sequential if the `combiner` method yields the exact same do-nothing instance as the one returned by the static `defaultCombiner` method. The factory methods handle this detail.

If you declare your own class, you need to be careful. A sequential gatherer should *not* override `combiner` but instead keep the default method, which returns `defaultCombiner()`. If you have a parallelizable gatherer that doesn’t need a combiner, you need to declare one anyway.

Now let us look more closely at the `sampling` gatherer. The state holds the index of the element. The initializer constructs a new state instance.

The integrator is declared using the `Gatherer.Integrator.of` factory method. Whenever an element is processed, the index is incremented. The element is pushed when the index is divisible by the sampling count.

Note that the integrator returns a `boolean`. You should return `false` when the pipeline "short-circuits" and no longer wants to accept elements. This can happen for two reasons. Your gatherer might want to short-circuit on its own. Simply return `false` in that case. (In the `sampling` example, that never happens.) Alternatively, a downstream operation can short-circuit. Then the call to `push` returns `false`. The integrator needs to return that downstream status.



Note: The `Gatherer.Downstream` interface has a method `isRejecting`. It allows an integrator to check whether a value would be accepted before it embarks on a costly computation.

Do not use the `isRejecting` method for reporting the downstream status. You want to keep the integrator code as simple as possible. Then it is more likely that the just-in-time compiler will inline it.

For optimizing the performance of a stream pipeline, it is important to know whether it contains any short-circuiting operations. An integrator that doesn't short-circuit should signal that fact by implementing the `Gatherer.Integrator.Greedy` interface. There is a factory method for that:

```
<T> Gatherer<T, IntermediateState, T> sampling(int n) {
    return Gatherer.ofSequential(
        IntermediateState::new,
        Gatherer.Integrator.ofGreedy((state, e, downstream) -> {
            state.index++;
            if (state.index % n == 0)
                return downstream.push(e);
            else
                return true;
            }));
}
```



Tip: You could create a class that implements the interface, but using a factory method is easier.



Tip: The `Gatherer.Integrator.of` method that you saw in the first implementation was not strictly necessary. We could have just used a lambda expression. It would have been converted to a `Gatherer.Integrator`. But the factory method tells the reader more clearly that this is not a greedy integrator.

As you have seen, writing your own gatherer may seem a bit daunting at first, but it is not all that complex. Follow these steps:

1. Decide whether your gatherer can execute in parallel (`Gatherer.of`), or whether it must see the elements sequentially (`Gatherer.ofSequential`).

2. Decide whether the integrator may short-circuit (`Gatherer.Integrator.of`) or if it always accepts elements (`Gatherer.Integrator.ofGreedy`).
3. Do you need state? If so, provide a class for the state, and if your gatherer is not sequential, implement a combiner.
4. In the integrator, return `false` when short-ciruiting, or when the downstream push is rejected.



Note: Unlike collectors, gatherers don't use enumerations for their characteristics. They use these alternative mechanisms:

1. The integrator of a greedy gatherer implements `Gatherer.Integrator.Greedy`. Otherwise the gatherer is short-circuiting.
2. The combiner of a sequential gatherer is the object returned by `Gatherer.defaultCombiner()`. Otherwise the gatherer is parallelizable.
3. The initializer of a stateless gatherer is the object returned by `Gatherer.defaultInitializer()`. Otherwise the gatherer is stateful.

***java.util.Stream<T>* 8**

- `<R> Stream<R> gather(Gatherer<? super T, ?, R> gatherer)` **24**
  yields a stream that results from processing the elements of this stream with the given gatherer.

***java.util.stream.Gatherers* 24**

- `static <TR> Gatherer<TR,?,List<TR>> windowFixed(int windowSize)`
  `static <TR> Gatherer<TR,?,List<TR>> windowSliding(int windowSize)`
  yields a gatherer that groups stream elements into separate or overlapping windows of the given size.
- `static <T,R> Gatherer<T,?,R> fold(Supplier<R> initial, BiFunction<? super R, ? super T, ? extends R> folder)`
  `static <T,R> Gatherer<T,?,R> scan(Supplier<R> initial, BiFunction<? super R, ? super T, ? extends R> scanner)`
  yield a gatherer that repeatedly applies the binary operation to the preceding result and the next stream element. `fold` produces only the final result, while `scan` yields all intermediate results.
- `static <T,R> Gatherer<T,?,R> mapConcurrent(int maxConcurrency, Function<? super T, ? extends R> mapper)`
  yields a gatherer that applies the given function, running the computations in at most `maxConcurrency` virtual threads, producing a stream of the results.

***java.util.stream.Gatherer.Downstream<T>* 24**

- `boolean push(T element)`
  pushes the element to the next stage of the processing pipeline. Returns `false` if the element was rejected.

- `boolean isRejecting()`
  returns `true` if the processing pipeline no longer accepts elements.

java.util.stream.Gatherer<T, A, R> 24

- `Supplier<A> initializer()`
  `Gatherer.Integrator<A,T,R> integrator()`
  `BinaryOperator<A> combiner()`
  `BiConsumer<A, Gatherer.Downstream<? super R>> finisher()`
  return the initializer, integrator, combiner, and finisher of this gatherer.
- `static <T,R> Gatherer<T,Void,R> of(Gatherer.Integrator<Void,T,R> integrator)`
  `static <T,R> Gatherer<T,Void,R> of(Gatherer.Integrator<Void,T,R> integrator, BiConsumer<Void, Gatherer.Downstream<? super R>> finisher)`
  `static <T,A,R> Gatherer<T,A,R> of(Supplier<A> initializer, Gatherer.Integrator<A,T,R> integrator, BinaryOperator<A> combiner, BiConsumer<A, Gatherer.Downstream<? super R>> finisher)`
  yield a parallelizable gatherer with the given initializer, integrator, combiner, and finisher. When there is no initializer, the gatherer is stateless.
- `static <T,R> Gatherer<T,Void,R> ofSequential(Gatherer.Integrator<Void,T,R> integrator)`
  `static <T,A,R> Gatherer<T,A,R> ofSequential(Supplier<A> initializer, Gatherer.Integrator<A,T,R> integrator)`
  `static <T,R> Gatherer<T,Void,R> ofSequential(Gatherer.Integrator<Void,T,R> integrator, BiConsumer<Void, Gatherer.Downstream<? super R>> finisher)`
  `static <T,A,R> Gatherer<T,A,R> ofSequential(Supplier<A> initializer, Gatherer.Integrator<A,T,R> integrator, BiConsumer<A, Gatherer.Downstream<? super R>> finisher)`
  yield a sequential gatherer with the given initializer, integrator, and finisher. When there is no initializer, the gatherer is stateless.

java.util.stream.Gatherer.Integrator<T> 24

- `boolean integrate(A state, T element, Gatherer.Downstream<? super R> downstream)`
  This method is called for each stream element. Update the state, and push a result downstream when appropriate. Return `false` if further elements will be rejected.
- `static <A,T,R> Gatherer.Integrator<A,T,R> of(Gatherer.Integrator<A,T,R> integrator)`
  `static <A,T,R> Gatherer.Integrator.Greedy<A,T,R> ofGreedy(Gatherer.Integrator.Greedy<A,T,R> greedy)`
  yield a short-circuiting or greedy integrator. The argument is intended to be a lambda expression.

1.12. Primitive Type Streams

So far, we have collected integers in a `Stream<Integer>`, even though it is clearly inefficient to wrap each integer into a wrapper object. The same is true for the other primitive types—`double`, `float`, `long`, `short`, `char`, `byte`, and `boolean`. The stream library has specialized types `IntStream`, `LongStream`, and `DoubleStream` that store primitive values directly, without using wrappers. If you want to store `short`, `char`, `byte`, and `boolean`, use an `IntStream`; for `float`, use a `DoubleStream`.

To create an IntStream, call the IntStream.of and Arrays.stream methods:

```
IntStream stream = IntStream.of(1, 1, 2, 3, 5);
stream = Arrays.stream(values, from, to); // values is an int[] array
```

As with object streams, you can also use the static generate and iterate methods. In addition, IntStream and LongStream have static methods range and rangeClosed that generate integer ranges with step size one:

```
IntStream zeroToNinetyNine = IntStream.range(0, 100); // Upper bound is excluded
IntStream zeroToHundred = IntStream.rangeClosed(0, 100); // Upper bound is included
```

The CharSequence interface has methods codePoints and chars that yield an IntStream of the Unicode codes of the characters or of the code units in the UTF-16 encoding. (See Chapter 2 for the sordid details.)

```
String greeting = "Ahoy 🏴‍☠️";
IntStream codes = greeting.codePoints();
    // The stream with values 67, 105, 97, 111, 32, 127988, 8205, 9760, 65039
```

The RandomGenerator interface has methods ints, longs, and doubles that return primitive type streams of random numbers.

```
IntStream randomIntegers = RandomGenerator.getDefault().ints();
```

When you have a stream of objects, you can transform it to a primitive type stream with the mapToInt, mapToLong, or mapToDouble methods. For example, if you have a stream of strings and want to process their lengths as integers, you might as well do it in an IntStream:

```
Stream<String> words = . . .;
IntStream lengths = words.mapToInt(String::length);
```



Caution: When converting a Stream<Integer> to an IntStream, use

```
numbers.mapToInt(i -> i)
```

and not

```
numbers.mapToInt(Function.identity()) // ERROR
```

The latter is a compile-time error, since there is no automatic unboxing.

To convert a primitive type stream to an object stream, use the boxed method:

```
Stream<Integer> integers = IntStream.range(0, 100).boxed();
```

Generally, the methods on primitive type streams are analogous to those on object streams. Here are the most notable differences:

- The toArray methods return primitive type arrays.

- Methods that yield an optional result return an `OptionalInt`, `OptionalLong`, or `OptionalDouble`. These classes are analogous to the `Optional` class, but they have methods `getAsInt`, `getAsLong`, and `getAsDouble` instead of the `get` method.
- There are methods `sum`, `average`, `max`, and `min` that return the sum, count, average, maximum, and minimum. These methods are not defined for object streams.
- The `summaryStatistics` method yields an object of type `IntSummaryStatistics`, `LongSummaryStatistics`, or `DoubleSummaryStatistics` that can simultaneously report the sum, count, average, maximum, and minimum of the stream.

The program in Listing 1.9 gives examples for the API of primitive type streams.

Listing 1.9: `v2ch01/streams/PrimitiveTypeStreams.java`

```
package v2ch01.streams;

import module java.base;

class PrimitiveTypeStreams {
    void main() throws Exception {
        IntStream is1 = IntStream.generate(() -> (int) (Math.random() * 100));
        show("is1", is1);
        IntStream is2 = IntStream.range(5, 10);
        show("is2", is2);
        IntStream is3 = IntStream.rangeClosed(5, 10);
        show("is3", is3);

        String contents = Files.readString(Path.of("gutenberg", "alice30.txt"));

        Stream<String> words = Stream.of(contents.split("\\PL+"));
        IntStream is4 = words.mapToInt(String::length);
        show("is4", is4);
        String sentence = "\uD835\uDD46 is the set of octonions.";
        IO.println(sentence);
        IntStream codes = sentence.codePoints();
        IO.println(codes.mapToObj(c -> "%X ".formatted(c)).collect(Collectors.joining()));

        Stream<Integer> integers = IntStream.range(0, 100).boxed();
        IntStream is5 = integers.mapToInt(Integer::intValue);
        show("is5", is5);
    }

    void show(String title, IntStream stream) {
        final int SIZE = 10;
        int[] firstElements = stream.limit(SIZE + 1).toArray();
        IO.print(title + ": ");
        for (int i = 0; i < firstElements.length; i++) {
            if (i > 0) IO.print(", ");
            if (i < SIZE)
                IO.print(firstElements[i]);
            else
                IO.print("...");
        }
        IO.println();
    }
}
```

***java.util.stream.IntStream* 8**

- `static IntStream range(int startInclusive, int endExclusive)`
- `static IntStream rangeClosed(int startInclusive, int endInclusive)`
  yield an `IntStream` with the integers in the given range.
- `static IntStream of(int... values)`
  yields an `IntStream` with the given elements.
- `int[] toArray()`
  yields an array with the elements of this stream.
- `int sum()`
- `OptionalDouble average()`
- `OptionalInt max()`
- `OptionalInt min()`
- `IntSummaryStatistics summaryStatistics()`
  yield the sum, average, maximum, or minimum of the elements in this stream, or an object from which all four of these values can be obtained.
- `Stream<Integer> boxed()`
  yields a stream of wrapper objects for the elements in this stream.

***java.util.stream.LongStream* 8**

- `static LongStream range(long startInclusive, long endExclusive)`
- `static LongStream rangeClosed(long startInclusive, long endInclusive)`
  yield a `LongStream` with the integers in the given range.
- `static LongStream of(long... values)`
  yields a `LongStream` with the given elements.
- `long[] toArray()`
  yields an array with the elements of this stream.
- `long sum()`
- `OptionalDouble average()`
- `OptionalLong max()`
- `OptionalLong min()`
- `LongSummaryStatistics summaryStatistics()`
  yield the sum, average, maximum, or minimum of the elements in this stream, or an object from which all four of these values can be obtained.
- `Stream<Long> boxed()`
  yields a stream of wrapper objects for the elements in this stream.

***java.util.stream.DoubleStream* 8**

- `static DoubleStream of(double... values)`
  yields a `DoubleStream` with the given elements.
- `double[] toArray()`
  yields an array with the elements of this stream.

- `double sum()`
- `OptionalDouble average()`
- `OptionalDouble max()`
- `OptionalDouble min()`
- `DoubleSummaryStatistics summaryStatistics()`
  yield the sum, average, maximum, or minimum of the elements in this stream, or an object from which all four of these values can be obtained.
- `Stream<Double> boxed()`
  yields a stream of wrapper objects for the elements in this stream.

***java.lang.CharSequence* 1.0**

- `IntStream codePoints()` **8**
  yields a stream of all Unicode code points.

java.util.random.RandomGenerator 17

- `IntStream ints()`
- `IntStream ints(int randomNumberOrigin, int randomNumberBound)`
- `IntStream ints(long streamSize)`
- `IntStream ints(long streamSize, int randomNumberOrigin, int randomNumberBound)`
- `LongStream longs()`
- `LongStream longs(long randomNumberOrigin, long randomNumberBound)`
- `LongStream longs(long streamSize)`
- `LongStream longs(long streamSize, long randomNumberOrigin, long randomNumberBound)`
- `DoubleStream doubles()`
- `DoubleStream doubles(double randomNumberOrigin, double randomNumberBound)`
- `DoubleStream doubles(long streamSize)`
- `DoubleStream doubles(long streamSize, double randomNumberOrigin, double randomNumberBound)`
  yield streams of random numbers. If `streamSize` is provided, the stream is finite with the given number of elements. When bounds are provided, the elements are between `randomNumberOrigin` (inclusive) and `randomNumberBound` (exclusive).
- `DoubleStream equiDoubles(double left, double right, boolean isLeftIncluded, boolean isRightIncluded)` **22**
  yields a stream of floating-point numbers between `left` and `right`, uniformly drawn from the largest possible sequence of equidistant numbers in the target interval. In contrast, the `doubles` methods draw uniformly from floating-point numbers in the interval between 0 and 1, then transforming into the target interval. That transformation can introduce a slight distortion.

java.util.OptionalInt 8

java.util.OptionalLong 8

java.util.OptionalDouble 8

- `static Optional`*`Xxx`*` of(`*`xxx`*` value)`
  yields an optional object with the supplied primitive type value.
- *`xxx`*` getAs`*`Xxx`*`()`
  yields the value of this optional object, or throws a `NoSuchElementException` if it is empty.
- *`xxx`*` orElse(`*`xxx`*` other)`
- *`xxx`*` orElseGet(`*`Xxx`*`Supplier other)`
  yield the value of this optional object, or the alternative value if this object is empty.
- `void ifPresent(`*`Xxx`*`Consumer consumer)`
  If this optional object is not empty, passes its value to `consumer`.

java.util.IntSummaryStatistics 8

java.util.LongSummaryStatistics 8

java.util.DoubleSummaryStatistics 8

- `long getCount()`
- *`xxx`*` getSum()`
- `double getAverage()`
- *`xxx`*` getMax()`
- *`xxx`*` getMin()`
  yield the count, sum, average, maximum, and minimum of the collected elements.

1.13. Parallel Streams

Streams make it easy to parallelize bulk operations. The process is mostly automatic, but you need to follow a few rules. First of all, you must have a parallel stream. You can get a parallel stream from any collection with the `Collection.parallelStream()` method:

```
Stream<String> parallelWords = words.parallelStream();
```

Moreover, the `parallel` method converts any sequential stream into a parallel one.

```
Stream<String> parallelWords = Stream.of(wordArray).parallel();
```

As long as the stream is in parallel mode when the terminal method executes, all intermediate stream operations will be parallelized.

When stream operations run in parallel, the intent is that the same result is returned as if they had run serially. It is important that the operations are *stateless* and can be executed in an arbitrary order.

Here is an example of something you cannot do. Suppose you want to count all short words in a stream of strings:

```
var shortWords = new int[12];
words.parallelStream().forEach(
    s -> { if (s.length() < 12) shortWords[s.length()]++; });
        // ERROR--race condition!
IO.println(Arrays.toString(shortWords));
```

This is very, very bad code. The function passed to `forEach` runs concurrently in multiple threads, each updating a shared array. As you saw in Chapter 10 of Volume I, that's a classic *race condition*. If you run this program multiple times, you are quite likely to get a different sequence of counts in each run—each of them wrong.

It is your responsibility to ensure that any functions you pass to parallel stream operations are safe to execute in parallel. The best way to do that is to stay away from mutable state. In this example, you can safely parallelize the computation if you group strings by length and count them:

```
Map<Integer, Long> shortWordCounts
    = words.parallelStream()
        .filter(s -> s.length() < 12)
        .collect(groupingBy(
            String::length,
            counting()));
```

By default, streams that arise from ordered collections (arrays and lists), from ranges, generators, and iterators, or from calling `Stream.sorted`, are *ordered*. Results are accumulated in the order of the original elements, and are entirely predictable. If you run the same operations twice, you will get exactly the same results.

Ordering does not preclude efficient parallelization. For example, when computing `stream.map(fun)`, the stream can be partitioned into *n* segments, each of which is concurrently processed. Then the results are reassembled in order.

Some operations can be more effectively parallelized when the ordering requirement is dropped. By calling the `Stream.unordered` method, you indicate that you are not interested in ordering. One operation that can benefit from this is `Stream.distinct`. On an ordered stream, `distinct` retains the first of all equal elements. That impedes parallelization—the thread processing a segment can't know which elements to discard until the preceding segment has been processed. If it is acceptable to retain *any* of the unique elements, all segments can be processed concurrently (using a shared set to track duplicates).

You can also speed up the `limit` method by dropping ordering. If you just want any `n` elements from a stream and you don’t care which ones you get, call

```
Stream<String> sample = words.parallelStream().unordered().limit(n);
```

As discussed in Section 1.9.2, merging maps is expensive. For that reason, the `Collectors.groupingByConcurrent` method uses a shared concurrent map. To benefit from parallelism, the order of the map values will not be the same as the stream order.

```
Map<Integer, List<String>> result = words.parallelStream().collect(
    Collectors.groupingByConcurrent(String::length));
    // Values aren't collected in stream order
```

Of course, you won't care if you use a downstream collector that is independent of the ordering, such as

```
Map<Integer, Long> wordCounts
    = words.parallelStream()
        .collect(
            groupingByConcurrent(
                String::length,
                counting()));
```

Don't turn all your streams into parallel streams in the hope of speeding up operations. Keep these issues in mind:

- There is a substantial overhead to parallelization that will only pay off for very large data sets.
- Parallelizing a stream is only a win if the underlying data source can be effectively split into multiple parts.
- The thread pool that is used by parallel streams can be starved by blocking operations such as file I/O or network access.

Parallel streams work best with huge in-memory collections of data and computationally intensive processing.



Tip: If you process the lines of a huge file, parallelizing the stream may improve performance. The `Files.lines` method uses a memory-mapped file, to make splitting effective. (If you simply read a line at a time, then you would have to read the first half of the file before the second half, and parallelization would not work.)



Note: If you need random numbers in parallel streams, obtain a generator from the `RandomGenerator.SplittableGenerator.of` factory method.



Note: By default, parallel streams use the global fork-join pool returned by `ForkJoinPool.commonPool`. That is fine if your operations don't block and you don't share the pool with other tasks. There is a trick to substitute a different fork-join pool. Execute the parallel stream operations in a thread from that pool:

```
ForkJoinPool customPool = . . .;
result = customPool.submit(() ->
    stream.parallel().map(. . .).collect(. . .)).get();
```

Or, asynchronously:

```
CompletableFuture.supplyAsync(() ->
        stream.parallel().map(. . .).collect(. . .), customPool)
    .thenAccept(result -> . . .);
```

This only works with a fork-join pool, not with arbitrary executors.

The example program in Listing 1.10 demonstrates how to work with parallel streams.

Listing 1.10: v2ch01/parallel/ParallelStreams.java

```
1  package v2ch01.parallel;
2
3  import static java.util.stream.Collectors.*;
4
5  import module java.base;
6
7  class ParallelStreams {
8      void main() throws Exception {
9          String contents = Files.readString(Path.of("gutenberg", "alice30.txt"));
10         List<String> wordList = List.of(contents.split("\\PL+"));
11
12         // Very bad code ahead
13         var shortWords = new int[10];
14         wordList.parallelStream().forEach(s -> {
15             if (s.length() < 10) shortWords[s.length()]++;
16         });
17         IO.println(Arrays.toString(shortWords));
18
19         // Try again--the result will likely be different (and also wrong)
20         Arrays.fill(shortWords, 0);
21         wordList.parallelStream().forEach(s -> {
22             if (s.length() < 10) shortWords[s.length()]++;
23         });
24         IO.println(Arrays.toString(shortWords));
25
26         // Remedy: Group and count
27         Map<Integer, Long> shortWordCounts = wordList.parallelStream()
28                 .filter(s -> s.length() < 10).collect(groupingBy(String::length, counting()));
29
30         IO.println(shortWordCounts);
31
32         // Downstream order not deterministic
33         Map<Integer, List<String>> result = wordList.parallelStream()
34                 .collect(Collectors.groupingByConcurrent(String::length));
35
36         IO.println(result.get(14));
37
38         result = wordList.parallelStream()
39                 .collect(Collectors.groupingByConcurrent(String::length));
40
41         IO.println(result.get(14));
42
43         Map<Integer, Long> wordCounts = wordList.parallelStream()
44                 .collect(groupingByConcurrent(String::length, counting()));
```

```
45
46          IO.println(wordCounts);
47      }
48  }
```

***java.util.stream.BaseStream<T,S extends BaseStream<T,S>>* 8**

- `S parallel()`
  yields a parallel stream with the same elements as this stream.
- `S unordered()`
  yields an unordered stream with the same elements as this stream.

***java.util.Collection<E>* 1.2**

- `Stream<E> parallelStream()` **8**
  yields a parallel stream with the elements of this collection.

In this chapter, you have learned how to put the stream library to use. The next chapter covers another important topic: processing input and output.

This page intentionally left blank

Index

Symbols

method
  of LocalTime 327
≤ operator 365
!= operator (SQL) 269
". . .", in XML 151
(number sign)
  in decimal formatting 349
  in message formatting 364, 365
  in URLs 232
#PCDATA (DTDs) 164
$ (dollar sign)
  in native method names 742
  in regular expressions 135, 136, 139, 145
% (percent sign)
  in locales 346
  in SQL 269, 292
  in URLs 240
& (ampersand)
  in CDATA sections 153
  in entity references 153
  parsing 165
&#, &#x, in character references 153
&& operator 137
&, entity reference 153
', entity reference 153
>, entity reference 153
<, entity reference 153
", entity reference 153
> (right angle bracket) 153
< (left angle bracket)
  in CDATA sections 153
  in message formatting 365
  parsing 165
<!--. . .-->, <?. . .?>, <![CDATA[. . .]]>, in XML 153
<> operator (SQL) 269
<. . .> (angle brackets, in regular expressions) 138
' 153
'. . .', in SQL 269
(. . .) (parentheses)
  in regular expressions 135, 138, 142
* (asterisk)
  echo character 510
  in DTDs 164, 166
  in glob patterns 97
  in regular expressions 135, 138, 143
+ (plus sign)
  in DTDs 164, 165
  in regular expressions 135, 138
  in URLs 240
  operator, identity for 39
, (comma)
  decimal 340, 346, 352
  in DTDs 164, 165
- (minus sign)
  in regular expressions 135
  in URLs 240
. (period)
  decimal 340, 346, 352
  in regular expressions 135, 136, 146
  in URLs 240
.., in paths 88
.class extension 398
/ (slash)
  in paths 64, 87
  in URLs 232
1931 CIE XYZ color specification 698
2D graphics. See Java 2D API
2D shapes 454
: (colon)
  in classpath 272
  in text files 71
  in URLs 232
; (semicolon)
  in classpath 272
  in decimal formatting 349
  in SQL 273
= operator (SQL) 269
? (question mark)
  in DTDs 164, 166
  in glob patterns 97
  in prepared queries 285
  in regular expressions 135, 136, 138
  in URLs 239
@ (at sign)
  in URIs 232
  in XPath 179
@Override annotation 127
@Resource annotation 315
@Serial annotation 125, 126, 127
@SuppressWarnings annotation 131
[(array), type code 119
[. . .] (brackets)
  in DOCTYPE declaration 163
  in glob patterns 97
  in regular expressions 135, 137
  in XPath 178
\ (backslash)
  in glob patterns 97
  in paths 64, 87
  in regular expressions 135, 136, 145
\0, in regular expressions 136
\\|, in regular expressions 72
\a, \A, in regular expressions 136
\A, in regular expressions 139
\b, \B, in regular expressions 139
\d, \D, in regular expressions 137
\e, \E, in regular expressions 136, 137
\f (form feed character literal) 136

\G, in regular expressions 139
\h, \H, in regular expressions 137
\k, in regular expressions 138
\n (newline character literal)
  as line feed 70, 158, 259, 367
  in regular expressions 136, 138, 146
\p, \P, in regular expressions 137
\Q 137
\r (carriage return character literal) 70, 158, 367
  in e-mails 259
\r, \R, in regular expressions 136, 139
\s, \S, in regular expressions 137
\t (tab character literal) 136
\u (Unicode character literal) 136
\v, \V, in regular expressions 137
\w, \W, in regular expressions 137
\x, in regular expressions 136
\z, \Z, in regular expressions 139
^ (caret) 135, 139, 146
_ (underscore)
  in native method names 742, 743
  in SQL 269, 292
  in URLs 240
{. . .} (braces)
  in glob patterns 97
  in message formatting 362, 365
  in regular expressions 135, 138, 145
| operator
  in DTDs 164, 166
  in message formatting 364
  in regular expressions 135, 138
  in text files 71
~ (tilde), in URLs 240
¤ (currency sign) 350
€ 352, 367

A

abort method
  of LoginModule 418
absolute method
  of ResultSet 296, 299
AbstractAction class 478, 481, 531, 534
  Constructor 534
AbstractButton class 488, 519, 533, 534, 535, 538
  isSelected method 535
  setAction method 533
  setActionCommand method 519
  setDisplayedMnemonicIndex method 537, 538
  setHorizontalTextPosition method 533, 534
  setMnemonic method 538
  setSelected method 535
AbstractCellEditor class 607, 608
  isCellEditable method 607
AbstractTableModel class 585
  getColumnName method 586
  isCellEditable method 604
Accelerators (in menus) 537, 538
Accept 218, 221
accept method
  of FileFilter 574, 578
  of FileFilter (Swing) 574
  of ServerSocket 221
acceptChanges method 302
  of CachedRowSet 303
Accessory components 576
Accumulator functions 39
accumulator method
  of Collector 38
Action interface 477, 481, 531
  actionPerformed method 477
  add/removePropertyChangeListener methods 477, 478
  get/putValue methods 477
  getValue method 481
  is/setEnabled methods 477
  isEnabled method 481
  predefined action table names in 478
  putValue method 481
  setEnabled method 481
Action listeners 477
ActionEvent class 469, 488
  getActionCommand, getModifiers methods 488
ActionListener interface
  actionPerformed method 470, 477, 478, 488
    overriding 531
  implementing 470
ActionMap class 480
Actions 477
  associating with keystrokes 479
  names of 481
  predefined 478
Adapter classes 475
add method
  of Area 663, 664
  of AttributeSet 734, 738
  of ButtonGroup 519
  of Container 470, 474, 503
  of DefaultMutableTreeNode 616, 622
  of JFrame 453
  of JMenu 531, 532
  of JToolBar 544, 546
addAttribute method
  of AttributesImpl 210
addBatch method 311
  of Statement 313
addCellEditorListener method
  of CellEditor 613
addChoosableFileFilter method
  of JFileChooser 578
addColumn method 596
  of JTable 600
addItem method 523
  of JComboBox 524
addLayoutComponent method
  of LayoutManager 558
addPropertyChangeListener method 477, 478
addRecipient method 260
AddressLayout interface 765
  withTargetLayout method 765
addSeparator method
  of JMenu 531, 532
  of JToolBar 544, 546
addTreeModelListener method 641
  of TreeModel 647
addTreeSelectionListener method 634
addURLs method 399
AdjustmentEvent class 488
AdjustmentListener interface 488

Adleman, Leonard 423, 438
AES 432
  generating keys in 433
Affine transformations 676, 703
AffineTransform class 676, 677
  Constructor 677
  getRotateInstance method 677
  getScaleInstance method 677
  getShearInstance method 677
  getTranslateInstance method 677
  get*Xxx*Instance methods 676, 679
  setToRotation method 678
  setToScale method 678
  setToShear method 678
  setToTranslation method 678
  setTo*Xxx* methods 676
AffineTransformOp class 703, 709
  Constructor 709
  TYPE_*XXX* fields 703
afterLast method 296
  of ResultSet 299
Aliases, for namespaces 170, 183
allMatch method 14
  of Stream 14
allocate method
  of ByteBuffer 107, 108
  of SegmentAllocator 750
allocateFrom method
  of SegmentAllocator 750
Alpha channel 680, 683
AlphaComposite class 682, 688
  getInstance method 683, 688
Alt+F4 538
anchor field
  of GridBagConstraints 554
Anchor rectangles 672
andFilter method 595
  of RowFilter 602
ANY (DTDs) 164
anyMatch method 14
  of Stream 14
Apache 149
  Derby 271
    connecting to 274
    drivers for 272
    populating 281
    starting 272
  DOM parser 154
  Tomcat 315
API
  Foreign Functions and Memory 741
Apollo 11, launch of 321, 328
append method
  of Appendable 63
  of JTextArea 513
  of Path2D 655, 663, 679
Appendable interface 62, 63
  append method 63
appendChild method 193
  of Node 195
Applets
  executing 418, 430
  security mechanisms in 397
Application servers 315
Applications
  business logic vs. visual representation of 265, 266
  client/server 265
  closing by user 446
  configuring 149, 150
  enterprise 314
  executing
    loading classes for 398
  localizing 369
  paid 403
  server-side 239
  signing 430
  web 314
Apply-templates 203
applyPattern method
  of MessageFormat 363
Arc2D class 650, 652
Arc2D.Double class 651, 662
  Constructor 662
Arc2D.Float class 651
ArcMaker 656
Arcs
  bounding rectangles of 651, 652
  closure types of 652
  computing angles of 652
Area class 664
  add method 663, 664
  exclusiveOr method 664
  intersect method 664
  subtract method 664
Arena interface 747, 748, 749
  automatic 748
  confined 748
  global 748
  ofAuto method 749
  ofConfined method 749
  ofShared method 749
  shared 748
ARGB 683, 699
ARRAY 313, 314
Arrays
  converting to/from streams 3, 22, 52
  getting from a database 314
  of strings 144
  type code for 119
Arrays class 8
  stream method 3, 8, 47
Ascender, ascent (in typesetting) 463
asCharBuffer method
  of ByteBuffer 107
ASCII 74, 146
  in property files 371
  native code sand 744
asMatchPredicate method 140
asPredicate method 140
Associative operations 39
Atomic operations 92
ATTLIST (DTDs) 166
attribute (XML Schema) 172
Attribute interface 731, 738
  getCategory method 733, 738
  getName method 738
  implementing 733
Attribute sets 733

Attributes interface 189
  enumerating 158
  getLength method 189
  getLocalName method 189
  getQName method 189
  getURI method 189
  getValue method 189
  in XML Schema 172
  legal 166
  names of 151
  namespace of 183
  values of 151
    accessing in XPath 179
    copying with XSLT 203
    default (DTDs) 167
    normalizing 167
  vs. elements 152, 153, 167, 198
AttributeSet interface 732, 738
  add method 734, 738
  get method 734, 738
  remove method 738
  toArray method 738
AttributesImpl class 210
  addAttribute method 210
  clear method 210
atZone method 328
Authentication 409, 427
  role-based 412
  through a trusted intermediary 427, 428
Autoboxing 582
AutoCloseable interface
  close method 62
Autocommit mode 310, 312
Autoflushing 70
Automatic arena 748
available method
  of InputStream 58, 59
availableCharsets method 75
availableLocales method
  of Locale 27
average method
  of DoubleStream 50
  of IntStream 49
  of LongStream 49
average method (primitive streams) 48
averagingXxx methods (Collectors) 31
AWT 443, 647
  event hierarchy in 486
  preferred field sizes in 507
AWTEvent class 486

B

B (byte), type code 119
Background color
  default 447
  setting 461, 472
Banding 712
Banner
  getPageCount method 719
  layoutPages method 719
Banners, printing 718
Base classes. See Superclasses
Baseline 463
BaseStream interface 22, 55
  iterator method 22
  parallel method 55
  unordered method 55
BasicButtonUI 501
BasicFileAttributes interface 94, 95
  creationTime method 95
  fileKey method 95
  isDirectory method 95
  isOther method 95
  isRegularFile method 95
  isSymbolicLink method 95
  lastAccessTime method 95
  lastModifiedTime method 95
  size method 95
BasicStroke class 664, 671
  Constructor 671
Batch updates 311
BCP 47 memo 341
beforeFirst method 296
  of ResultSet 299
between method
  of Duration 318, 320
Bevel join 665
Bicubic, bilinear interpolations 703
Big-endian order 74, 78, 368
Big5 366
BigDecimal class 126
Binary data
  converting to Unicode code units 67
  reading/writing 78, 79
  vs. text 67
Bindings interface 390, 391
  get method 391
  put method 391
Birthdays, calculating 322
BitSet class 39
BLOB 271, 313
Blob interface 290, 291
  getBinaryStream method 290, 291
  getBytes method 290, 291
  length method 291
  setBinaryStream method 291
BLOBs 290
  placing in database 290
Blocking
  by I/O methods 58
  by network connections 211, 215, 224
Blur filter 705
BMP 688
body method
  of HttpResponse 248, 254
BodyHandlers
  ofString method 248, 249
BodyPublishers class
  ofString method 248
Book class 718
BOOLEAN 271, 313
boolean type
  printing 70
  streams of 46
  type code for 119
  writing in binary format 78
Bootstrap class loader 398, 399

Border layout manager 504
BorderFactory class 520
  createBevelBorder method 521
  createCompoundBorder method 521
  createEmptyBorder method 520
  createEtchedBorder method 521
  createLineBorder method 520
  createLoweredBevelBorder method 521
  createMatteBorder method 520
  createRaisedBevelBorder method 521
  createTitledBorder method 521
BorderFactory class, methods of 519
BorderLayout class 504, 506
  Constructor 506
Borders 519
Bounding rectangle 456
Bounding rectangles 651
Box layout 546
boxed method
  of DoubleStream 50
  of IntStream 49
  of LongStream 49
boxed method (primitive streams) 47
Bray, Tim 151
Breadth-first enumerations 629, 630
breadthFirstEnumeration method 629
  of DefaultMutableTreeNode 633
Browsers
  forms in 239
  response pages in 239
Buffer class 106, 107, 109
  clear method 108, 109
  flip method 108, 109
  hasRemaining method 106
  limit method 106
  mark method 108, 109
  remaining method 108, 109
  reset method 108, 109
  rewind method 108, 109
BufferedImage class 672, 696, 701
  constructor 697, 701
  getColorModel method 698, 701
  getRaster method 697, 701
  TYPE_BYTE_GRAY field 699
  TYPE_INT_ARGB field 697, 698
BufferedImageOp interface 696, 709
  filter method 703, 709
  implementing 703
BufferedInputStream class 67
  Constructor 67
BufferedOutputStream class 67
  Constructor 67
Buffers 107
  capacity of 108
  flushing 58, 70
  in-memory 63
  limits of 108
  marks in 108
  positions in 102, 108
  traversing all bytes in 102
  vs. random access 102
build method
  of HttpClient.Builder 247, 248, 253
  of HttpRequest.Builder 254
Bundle classes 371
Butt cap 665
ButtonGroup class 516, 519
  add method 519
  getSelection method 517, 519
ButtonModel interface 499, 519
  getActionCommand method 517, 519
  getSelectedObjects method 517
  properties of 500
Buttons
  appearance of 498
  associating actions with 479
  clicking 472
  creating 470
  event handling for 470
  model-view-controller analysis of 499, 501
  rearranging automatically 502
ButtonUIListener 501
Byte order mark 75, 368
byte type
  streams of 46
  type code for 119
Byte-oriented input/output streams 57
ByteArrayClass 381
ByteArrayClassLoader 382
ByteArrayOutputStream class 133
ByteBuffer class 102, 106, 107, 109
  allocate method 107, 108
  asCharBuffer method 107
  compact method 109
  get method 106
  get, getXxx methods 103
  getXxx method 107
  order method 103, 107
  put method 106, 107
  put, putXxx methods 103
  putXxx method 107
  wrap method 107, 108
Bytecodes
  engineering
    with hex editor 408
  verifying 405
  writing to memory 381
ByteLookupTable class 704, 709
  Constructor 709
byteOffset method
  of MemoryLayout 753
byteOffsetHandle method
  of MemoryLayout 765
Bytes, reading/writing 57
Bézier curves 655

C

C
  calling
    from Java programs 742
  database access in 264
  FILE* type in 60
  pointers in 742
  strings in 744
C (char), type code 119
C++
  for native methods 744

pointers in 742
Cached row sets 300, 305
CachedRowSet interface 300, 302
acceptChanges method 302, 303
execute method 301, 302
getPageSize method 301, 303
getTableName method 302, 303
nextPage method 301, 303
populate method 301, 303
previousPage method 303
setPageSize method 301, 303
setTableName method 302, 303
Caesar cipher 401
Calendar class 317
formatting objects of 354
weekends in 322
call escape (SQL) 292
call method
of CompilationTask 380
of JavaCompiler.CompilationTask 386
Callable interface 380
Callback interface 413
foreign function 759
CallbackHandler interface 417
handle method 417
cancelCellEditing method 607, 608
of CellEditor 613
cancelRowUpdates method 297
of ResultSet 299
canInsertImage method 691
of ImageWriter 696
Canonical layouts 751
canonicalLayouts method
of Linker 758
captureCallState method
of Linker.Option 765
Carriage return character, displaying 158
catalog (XML catalogs) 164
CatalogFeatures class 170
defaults method 164, 170
CatalogManager class 170
catalogResolver method 164, 170
catalogResolver method
of CatalogManager 170
CDATA 166, 167
CDATA sections 153
Cell editors 604, 608
Cell renderers
for tables 589, 602, 604
for trees 631
CellEditor interface 613
addCellEditorListener method 613
cancelCellEditing method 607, 608, 613
getCellEditorValue method 605, 607, 608, 613
isCellEditable method 613
removeCellEditorListener method 613
shouldSelectCell method 607, 608, 613
stopCellEditing method 607, 608, 613
Cells (Swing)
editing 604
selecting 592
Certificates 409, 424
for software developers 430
Java Plug-in and 430
publishing fingerprints of 425
signing 429
Chain of trust 427
ChangeListener interface, stateChanged method 525
Channels class 224, 227
for files 102
newInputStream method 227
newOutputStream method 225, 227
char type
streams of 46
type code for 119
CHARACTER 271, 313
Character classes 135
Character encodings 67, 74
character order in 356
explicitly specified 75
of source files 368
partial 75
platform 76, 366
Character references 153
CharacterData interface 161
getData method 157, 161
characteristics method
of Collector 38
Characters
differences between 357
escaping 72
normalizing 358
outlines of 679
printing 70
writing in binary format 78
characters method
of ContentHandler 185, 189
charAt method
of CharSequence 63
CharBuffer class 63, 107
get method 107
put method 107
CharSequence interface 50, 63
charAt method 63
chars method 47
codePoints method 47, 50
length method 64
splitting 4
subSequence method 64
toString method 64
Charset class
availableCharsets method 75
defaultCharset method 76, 367
forName method 75
Checkboxes 514, 534, 602
checked attribute (HTML) 151
checkError method 70
of PrintWriter 71
Child elements 152
namespace of 182
Child nodes 614
adding 616
connecting lines for 618, 619
children method
of TreeNode 629
Choice components 513
checkboxes 514, 534
combo boxes 522

radio buttons 516, 534
sliders 525
choice element (XML Schema) 172
choice keyword (message formatting) 364
Church, Alonzo 321
Cipher class 431, 436
doFinal method 432, 434, 436, 437
getBlockSize method 436
getInstance method 431, 436
getOutputSize method 436
init method 436
update method 432, 434, 436, 437
*XXX*_MODE modes 432
CipherInputStream class 437
Constructor 437
read method 438
CipherOutputStream class 437, 438
Constructor 438
flush method 438
write method 438
Ciphers
Caesar 401
keys for
generating 433
public 422, 428, 438
performance of 438
streams for 437
symmetric 431, 438
Class class 404
forName method 383
getClassLoader method 398, 404
getFields method 642
Class files 398
corrupted 406, 408
encrypted 401
loading 398
portability of 368
verifying 405
Class loaders 398
as namespaces 401
bootstrap 398, 399
context 400
extension 399
hierarchy of 399
separate for each web page 401
specifying 399
system 399
writing 401
Class path, adding JAR files to 399
Classes
adapter 475
compiling on the fly 381
externalizable 119
generic 522
helper 550
inheritance trees of 630
loading 382, 399
nonserializable 125
platform 398
resolving 398
separate for each web page 401
serializable 113, 114, 125, 131
deserializing 132
static inner 121
versioning 129
Classifier functions 30
ClassLoader class 398, 405
defineClass method 401, 405
extending 382, 401
findClass method 401, 405
getParent method 405
getSystemClassLoader method 405
loadClass method 400, 401
Classloader inversion 400
CLASSPATH 399
CLEAR composition rule 682
clear method
of AttributesImpl 210
of Buffer 108, 109
clearParameters method
of PreparedStatement 289
Client/server applications 265
Clients
connecting to servers 213
multiple, serving 221
clip method
of Graphics2D 648, 678, 680, 713
Clipping shapes 647, 678
printing 713
setting region for 648
CLOB data type (SQL) 271, 313
Clob interface 290, 291
getCharacterStream method 290, 291
getSubString method 290, 291
length method 291
setCharacterStream method 291
CLOBs 290
placing in database 290
Clone 133
clone method
of Object 114
Cloning 133
close method
of AutoCloseable 62
of Closeable 62, 63
of Connection 277, 279, 315
of FileLock 113
of FileSystems 101
of Flushable 62
of InputStream 58, 59
of OutputStream 60
of ResultSet 278, 279
of ServerSocket 221
of Statement 278, 279
of XMLStreamWriter 201
Closeable interface 62, 63
close method 62, 63
flush method 62
closeEntry method
of ZipInputStream 85
of ZipOutputStream 85
of Zip*Xxx*Stream 84
closeOnCompletion method 279
of Statement 278
closePath method 655
of Path2D 663
Closure types 652
cmd shell (Windows) 367

Code units 47
in regular expressions 136
writing 81
Codebreakers, The (Kahn) 402
codePoints method 47
of CharSequence 50
Collation 356
CollationKey class 361
compareTo method 361
Collator class 357, 361
compare method 361
equals method 361
getAvailableLocales method 361
getCollationKey method 358, 361
getDecompositon method 361
getInstance method 361
getStrength method 361
setDecomposition method 361
setStrength method 361
collect method
of Stream 22, 25, 39, 40
collectingAndThen method 32
of Collectors 35
Collection interface 3, 55
parallelStream method 2, 3, 51, 55
stream method 2, 3
Collections class
iterating over elements of 1
sort method 357
vs. streams 2
Collector interface 25, 38
accumulator method 38
characteristics method 38
combiner method 38
finisher method 38
of method 38
supplier method 38
Collectors class 24, 25, 26, 29, 30, 34
averaging*Xxx* methods 31
collectingAndThen method 32, 35
composing 33
counting method 31, 34
downstream 31, 53
filtering method 32, 35
flatMapping method 32, 35
groupingBy method 30, 34
groupingByConcurrent method 30, 52
joining method 23, 24
mapping method 32, 35
maxBy method 35
maxBy, minBy methods 31
minBy method 35
partitioningBy method 30, 31, 33
reducing method 33
summarizingDouble method 24
summarizingInt method 24
summarizingLong method 24
summarizing*Xxx* methods 23, 32
summingDouble method 35
summingInt method 35
summingLong method 35
summing*Xxx* methods 31
teeing method 33
toCollection method 25, 27
toConcurrentMap method 28, 29
toList method 25, 26
toMap method 27, 29
toSet method 25, 26, 31
toUnmodifiableList method 26
toUnmodifiableMap method 29
toUnmodifiableSet method 26
Color choosers 606
Color class 460, 461, 671, 702
Constructor 461, 702
getRGB method 702
translating values into pixel data 699
Color space conversions 704
ColorConvertOp class 703, 704
ColorModel class 699, 702
getDataElements method 702
getRGB method 699, 702
Colors
background/foreground 461
changing 478
components of 680
composing 681, 683
interpolating 671
negating 704
predefined/custom 460
solid 647
Columns 507
Columns (databases)
accessing by number, in result set 277
names of 266
number of 304
Columns (Swing)
accessing 590
adding 596
detached 583
hiding 596
names of 586
rendering 588, 590
resizing 583, 590, 591
selecting 592
com.sun.security.auth.module package 410
combiner method
of Collector 38
Combo box editors 605
Combo boxes 522
adding items to 523
Comments 154
commit method
of Connection 310, 312
of LoginModule 418
commonPool method 53
compact method
of ByteBuffer 109
Comparable interface 12
compareTo method 594
Comparator interface 12, 357
Comparators class 593
compare method
of Collator 361
compareTo method
of CollationKey 361
of Comparable 594
of Instant 318
of String 356

Compilable interface 393
  compile method 393
CompilationTask 380
  call method 380
compile method
  of Compilable 393
  of Pattern 140, 146
CompiledScript class 394
  eval method 394
Compiler
  invoking 379
Complex types 171
complexType method 171
Component class 449, 453, 461, 486, 489, 509
  getBackground method 461
  getFont method 509
  getForeground method 461
  getPreferredSize method 452, 453
  getSize method 449
  inheritance hierarchies of 503
  isVisible method 449
  repaint method 451, 453
  setBackground method 461
  setBackground/Foreground methods 461
  setBounds method 449
  setBounds, setLocation methods 447
  setCursor method 486
  setForeground method 461
  setLocation method 449
  setSize method 449
  setVisible method 447, 449
  validate method 509
Components (in layout) 501
  classes for 445
  displaying information in 450
  labeling 509
  visibility of 447
Composite interface 682
Composition rules 647, 648, 680
Computer Graphics: Principles and Practice (Hughes et al.) 655, 698
concat method
  of Stream 11, 12
Confidential information 430
Configuration files 110, 133, 490
Confined arena 748
Confirmation dialogs 561
connect method
  of Socket 216
  of URLConnection 234, 235, 238
Connection interface 277, 281, 289, 291, 298, 308, 312
  close method 277, 279, 315
  commit method 310, 312
  createBlob method 291
  createBlob, createClob methods 290
  createClob method 291
  createStatement method 276, 277, 294, 295, 297, 298, 310, 311
  getAutoCommit method 312
  getMetaData method 303, 308
  getWarnings method 281
  prepareStatement method 285, 286, 289, 295, 298
  releaseSavepoint method 311, 312
  rollback method 310, 312
  setAutoCommit method 310, 312
  setSavepoint method 312
Connections
  closing 279
    using row sets after 300
  debugging 261
  pooling 315
  starting new threads 221
console method
  of System 367
Constructive area geometry 663
Constructors
  bypassing 132
  no-argument 116, 132
Container class 474, 501, 503
  add method 470, 474, 503
  setLayout method 503
Containers 501
Content pane 450
Content types 234
ContentHandler class 185, 186
  characters method 185
  startDocument, endDocument methods 185
  startElement, endElement methods 185
ContentHandler interface 188
  characters method 189
  endDocument method 188
  endElement method 189
  startDocument method 188
  startElement method 189
Context class loader 400
Control points
  dragging 656
  of curves 654
  of shapes 656
Controllers 498
convertColumnIndexToModel method
  of JTable 600
convertRowIndexToModel method
  of JTable 600
convert*Xxx*IndexToModel methods
  of JTable 592
Convolution operation 705
ConvolveOp class 703, 705, 706, 710
  Constructor 710
CookieHandler class
  setDefault method 244
Cookies 244
Coordinate system
  custom 648
  translating 714
Coordinate transformations 673
Copies 732, 734
  getValue method 734
CopiesSupported 732
copy method
  of Files 91, 93, 100
copyArea method
  of Graphics 469
CORBA 399
Core Swing (Topley) 581, 614, 625
count function
  of XPath 179
count method

of Stream 2, 3, 13
counting method
of Collectors 31, 34
Country codes 30, 341
CRC32 checksum 101, 103
CRC32 class 103
CREATE TABLE 270
executing 276
in batch updates 311
createBevelBorder method
of BorderFactory 521
createBindings method
of ScriptEngine 390
createBlob method
of Connection 291
createBlob, createClob methods
of Connection 290
createCachedRowSet method 300, 301
of RowSetFactory 303
createClob method
of Connection 291
createCompoundBorder method
of BorderFactory 521
createDirectories method
of Files 91
createDirectory method
of Files 91
createDirectory, createDirectories methods
of Files 90
createElement method
of Document 195
createElement, createElementNS methods
of Document 192
createElementNS method
of Document 195
createEmptyBorder method
of BorderFactory 520
createEtchedBorder method
of BorderFactory 521
createFile method 90
of Files 91
createFilteredRowSet method
of RowSetFactory 303
createImageInputStream method
of ImageIO 695
createImageOutputStream method
of ImageIO 695
createImage*Xxx*Stream methods of ImageIO 690
createJdbcRowSet method
of RowSetFactory 303
createJoinRowSet method
of RowSetFactory 303
createLineBorder method
of BorderFactory 520
createLink method
of Files 91
createLoweredBevelBorder method
of BorderFactory 521
createLSOutput method 194
createLSSerializer method 194
createMatteBorder method
of BorderFactory 520
createPrintJob method 728
of PrintService 728
createRaisedBevelBorder method
of BorderFactory 521
createServerSocket method
of ServerSocketFactory 230
createSocket method
of SocketFactory 231
createStatement method 276, 294, 295, 297, 310, 311
of Connection 277, 298
createSymbolicLink method
of Files 91
createTempDirectory method
of Files 91
createTempFile method
of Files 91
createTemp*Xxx* methods
of Files 91
createTextNode method 193
of Document 195
createTitledBorder method
of BorderFactory 521
create*Type*Border methods (BorderFactory) 519
createWebRowSet method
of RowSetFactory 303
createXMLStreamReader method
of XMLInputFactory 191
createXMLStreamWriter method 196
of XMLOutputFactory 200
creationTime method
of BasicFileAttributes 95
Credit card numbers 430
Cryptography and Network Security (Stallings) 420
Ctrl+O, Ctrl+S accelerators 537
Cubic curves 654
CubicCurve2D class 650, 654
CubicCurve2D.Double class 651, 663
Constructor 663
CubicCurve2D.Float class 651
Currencies 352
formatting 350
identifiers for 352
Currency class 352, 353
getAvailableCurrencies method 353
getCurrencyCode method 353
getDefaultFractionDigits method 353
getInstance method 352, 353
getNumericCode method 353
getNumericCodeAsString method 353
getSymbol method 353
toString method 353
Current user 490
Cursor class, getPredefinedCursor method 483
Cursor shapes 483
curveTo method 655
of Path2D.Float 663
Custom layout managers 555
Customizations. See Preferences
Cyclic references 132

D

D (double), type code 119
d (SQL escape) 292
Dashed lines 666
Data

encrypting/decrypting 437
fingerprints of 419
signed 422
Data exchange 566
Data sources (for JNDI service) 315
Data types
codes for 119
print services for 726
database.properties package 282, 314
DatabaseMetaData interface 300, 303, 308, 313
getJDBCMajorVersion method 308
getJDBCMinorVersion method 308
getMaxConnections method 309
getMaxStatements method 279, 309
getSQLStateType method 279
getTables method 304, 308
supportsBatchUpdates method 311, 313
supportsResultSetConcurrency method 300
supportsResultSetType method 300
supportsResultSet*Xxx* methods 295
Databases 263, 316
accessing, in C language 264
autocommit mode of 310, 312
batch updates for 311
caching prepared statements 286
changing data with SQL 270
connections to 272, 273, 282
closing 279, 282
in web and enterprise applications 314
pooling 315
drivers for 264
duplicating data in 267
error handling in 312
escape syntax in 291
integrity of 310
JAR files for 272
keys in 294
LOBs in 290
metadata for 303
modifying 301
native storage for XML in 314
numbering columns in 277
outer joins in 292
populating 281
registering classes for 274
scalar functions in 292
scrollable/updatable result sets in 295
setting up parameters in 301
starting 272
stored procedures in 292
structure of 266, 303
synchronization of 302
tools for 305
truncated data from 280
URLs of 272
DataInput interface 79
readBoolean method 79
readByte method 79
readChar method 79, 81
readDouble method 79, 114, 126
readFloat method 79
readFully method 80
readInt method 79, 81, 114
readLong method 79
readShort method 79
readUTF method 79, 80
skipBytes method 80
DataInputStream class 61, 64, 79
DataIO
*xxx*FixedString methods 81, 82
DataOutput interface 78, 80
writeBoolean method 78, 80
writeByte method 78, 80
writeChar method 78, 80, 81
writeChars method 78, 80
writeDouble method 78, 80, 114, 126
writeFloat method 78, 80
writeInt method 78, 80, 81, 114
writeLong method 78, 80
writeShort method 78, 80
writeUTF method 78, 80
DataOutputStream class 61, 79
DataSource interface 315
DataTruncation class 280, 281
getDataSize method 281
getIndex method 281
getParameter method 281
getTransferSize method 281
DATE 271, 292, 313
Date and Time API 317, 337
legacy code and 336
Date class 119, 126, 317, 336
formatting objects of 354
toInstant method 336
valueOf method 337
Date class
months and years in 321
dateFilter method 594
of RowFilter 602
DateFormat class 333, 337, 355
DateFormatter class 354
Dates
computing 322, 325
filtering 594
formatting 331, 340, 353, 361
literals for 291
local 321
nonexistent 354
parsing 334
datesUntil method 322
of LocalDate 324
DateTimeFormatter class 331, 335, 353, 356
format method 331, 335, 356
legacy classes and 336
ofLocalizedDate method 335, 356
ofLocalizedDateTime method 335, 356
ofLocalizedPattern method 336
ofLocalizedTime method 335, 356
ofLocalized*Xxx* methods 332, 353
ofPattern method 333, 336
parse method 334
toFormat method 333, 337
withLocale method 332, 336, 353, 356
DateTimeSyntax class 734
Daylight savings time 327
DayOfWeek enumeration 322
getDisplayName method 332, 354
dayOfWeekInMonth method

of TemporalAdjusters 326
Days of week 354, 355
DBeaver 305
Debugging
intermittent bugs 446
JDBC-related problems 274
mail connections 261
streams 12
with telnet 211
DECIMAL 271, 313
Decimal separators 340, 346, 352
DecimalFormat class 349
DecimalFormatSymbols class 351
decode method
of URLDecoder 247
Decomposition 358
DefaultButtonModel class 499, 501
DefaultCellEditor class 612, 626
Constructor 612
variations of 605
defaultCharset method 76, 367
DefaultComboBoxModel class 523
DefaultHandler class 186
defaultLookup method
of Linker 758
DefaultMutableTreeNode class 615, 622, 629, 632, 633
add method 616, 622
breadthFirstEnumeration method 633
Constructor 622
depthFirstEnumeration method 633
pathFromAncestorEnumeration method 630
postOrderEnumeration method 633
preOrderEnumeration method 633
setAllowsChildren method 621, 623
*xxx*FirstEnumeration methods 629
*xxx*OrderEnumeration methods 630
defaultPage method
of PrinterJob 717
DefaultRowSorter class 601
setComparator method 593, 601
setRowFilter method 594, 595, 601
setSortable method 593, 601
defaults method
of CatalogFeatures 164, 170
DefaultTableCellRenderer class 603
DefaultTableModel class
isCellEditable method 604
DefaultTreeCellRenderer class 631, 633
getTreeCellRendererComponent method 632
setClosedIcon method 633
setLeafIcon method 633
setOpenIcon method 633
set*Xxx*Icon methods 632
DefaultTreeModel class 615, 622, 623, 629, 642
automatic notifications by 625
getPathToRoot method 625
insertNodeInto method 624, 629
isLeaf method 621
nodeChanged method 625, 629
nodesChanged method 629
reload method 625, 629
removeNodeFromParent method 624, 629
setAsksAllowsChildren method 621, 622
defaultWriteObject method 126
defineClass method 401
of ClassLoader 405
DELETE 270
executing 276
in batch updates 311
vs. methods of ResultSet 298
delete method
of Files 92, 93
of HttpRequest.Builder 254
deleteIfExists method 92
of Files 93
deleteRow method 298
of ResultSet 299
Delimiters, in text files 71
Depth-first enumerations 629, 630
depthFirstEnumeration method 629
of DefaultMutableTreeNode 633
derbyclient.jar package 272
dereferenceElement method
of MemoryLayout.PathElement 765
DES 432
Descender, descent (in typesetting) 463
Deserialization 132
Device coordinates 673
Diagnostic interface 380, 386
getColumnNumber method 386
getKind method 386
getLineNumber method 386
getMessage method 386
getSource method 386
DiagnosticCollector class 380, 386
Constructor 386
getDiagnostics method 386
DiagnosticListener interface 380
Dialogs 558
accepting/canceling 567
closing 475, 476, 538, 565, 567
confirmation 561
creating 563
data exchange in 566
default button in 568
displaying 564
modal 559
modeless 559, 564, 565, 567
root pane of 568
Dictionary ordering 357
digest method
of MessageDigest 420, 421
DigiCert 425, 427
Digital fingerprints 119, 124, 419
computing 129
different for a class and its objects 122
Digital signatures 418
verifying 424
Directories
creating 90
current 99
hierarchical structure of 613
printing all subdirectories of 99
traversing 95
user's working 64
DirectoryStream interface 98
discarding method
of HttpResponse.BodyHandlers 249

distinct method
of Stream 12, 13, 52
dividedBy method
of Duration 321
Do-nothing methods 475
Doc interface 728
DocAttribute interface 731
implementing 733
printing attributes of 734, 738
DocAttributeSet interface 732, 733
DocFlavor class 726, 729
DocPrintJob interface 728, 738
getAttributes method 738
print method 728
DOCTYPE 163
including in output 194
Document flavors, for print services 726
Document interface 154, 161, 195
createElement method 195
createElementNS method 195
createTextNode method 195
create*Xxx* methods 192
getDocumentElement method 155, 161
DocumentBuilder class 161, 169, 194
newDocument method 192, 194, 206
parse method 161
setEntityResolver method 164, 169
setErrorHandler method 169
DocumentBuilderFactory class 160, 170, 185
isIgnoringElementContentWhitespace method 170
isNamespaceAware method 185
isValidating method 170
newDocumentBuilder method 154, 160, 193
newInstance method 154, 160, 183
newNSInstance method 184, 185, 187
setIgnoringElementContentWhitespace method 168, 170
setNamespaceAware method 173, 183, 185, 187, 193
setValidating method 168, 170
doFinal method 432, 434, 437
of Cipher 436
DOM (Document Object Model) parser 154, 185
namespace-awareness of 183, 187
trees in
accessing with XPath 178
analyzing 157, 158
building 185, 192, 206
writing 193
DOMImplementationLS
createLSOutput method 194
createLSSerializer method 194
DOMResult class 205, 210
Constructor 210
DOMSource class 196, 204
Constructor 196
DOUBLE 271, 313
double type
printing 70
streams of 46
type code for 119
writing in binary format 78
DoubleBuffer class 107
doubles method
of RandomGenerator 47, 50
DoubleStream interface 46, 49
average method 48, 50
boxed method 47, 50
mapToDouble method 47
max method 50
max, min methods 48
min method 50
of method 49
sum method 50
sum, summaryStatistics methods 48
summaryStatistics method 50
toArray method 47, 49
DoubleSummaryStatistics class 23, 25, 48, 51
getAverage method 25, 51
getCount method 25, 51
getMax method 25, 51
getMin method 25, 51
getSum method 25, 51
doubleValue method 346
Downcall handle 755
downcallHandle method
of Linker 759
Downstream collectors 31, 53
draw method
of Graphics2D 649, 650, 664
of Shape 454
drawImage method 467
of Graphics 469
Drawing with mouse 482
Drawings
creating 647
printing 710
drawString method
of Graphics2D 466
draw*Xxx* methods
of Graphics 650
DriverManager class 275
getConnection method 273, 275, 282, 315
setLogWriter method 274
DROP TABLE 274
executing 276
in batch updates 311
Drop-down lists 522
dropWhile method 11
of Stream 12
DSA 423
DST, DST_*Xxx* composition rules 682
DTDHandler interface 186
DTDs 162
element content in 164, 166
entities in 167
external 163
in XML documents 151, 163
locating 164
unambiguous 166
URLs for 163
Duration class 320
between method 318, 320
dividedBy method 321
getSeconds method 318
immutability of 319
isNegative method 321
isZero method 321
minus method 320
minus*Xxx* method 321

multipliedBy method 321
negated method 321
of*Xxx* method 320
plus method 320
plus*Xxx* method 321
toDaysPart method 320
toSeconds method 320
to*Xxx* method 320
to*Xxx* methods 318
to*Xxx*Part method 320
Dynamic web pages 383

E

E-mails
sending 259
terminating lines in 259
Echo character 510
Echo servers 220
Edge detection 705
ELEMENT (DTDs) 164, 166
element (XML Schema) 171
Element interface 161, 195
getAttribute method 158, 161
getTagName method 156, 161, 184
of Document 154
setAttribute method 195
setAttribute, setAttributeNS methods 193
setAttributeNS method 195
Elements
child 152
accessing in XPath 179
namespace of 182
constructing 192
counting, in XPath 179
empty 151
legal attributes of 166
mixed content in 152, 166
names of 156, 184
root 152, 170
stripping whitespace in 157
vs. attributes 152, 153, 167, 198
elements method
of SegmentAllocator 750
Ellipse2D class 454, 456, 650
setFrameFromCenter method 456
Ellipse2D.Double class 460
Constructor 460
Ellipse2D.Double, Ellipse2D.Float classes 651
Ellipses 454, 456, 651
bounding rectangles of 456
constructing 456
filling with color 460
EMPTY (DTDs) 164
empty method
of Optional 18
of Stream 4, 7
Empty tags 151
encode method
of URLEncoder 247
Encodings. See Character encodings
Encryption 430
decryption keys for 402
exporting strong methods of 402
final block padding in 432
of class files 401
End cap styles 665, 666
end method
of MatchResult 148
End points 654
End tags 151
End-of-line character. See Line feed
endDocument method 185
of ContentHandler 188
endElement method 185
of ContentHandler 189
Enterprise applications 314
Enterprise JavaBeans 265
Entity references 153, 167
Entity resolvers 155, 164
ENTITY, ENTITIES attribute types (DTDs) 166, 167
EntityResolver interface 169, 186
resolveEntity method 164, 169
Entrust 427
Entry 595
enumeration (XML Schema) 171
Enumeration interface
nextElement method 629
Enumerations
ignoring serialVersionUID fields 131
of nodes, in a tree 629
serialization of 128
using attributes for 167
EnumSyntax class 734
Epoch 126, 318
equals method
of Collator 361
of Instant 318
equiDoubles method
of RandomGenerator 50
errno
capturing 764
Error 169
Error handlers
installing 169
error method
of ErrorHandler 169
ErrorHandler interface 169, 186
error method 169
fatalError method 169
warning method 169
Errors
UnsatisfiedLinkError 742
Escapes
in regular expressions 72
in SQL 291
Essential XML (Box et al.) 149, 202
Euro symbol 352, 367
eval method
of CompiledScript 394
of ScriptEngine 389, 390
evaluate method
of XPath 181
evaluate, evaluateExpression methods (XPath) 179, 180
evaluateExpression method
of XPath 181
Event delegation model 469
Event dispatch thread 446

Event handling 469, 488
  semantic vs. low-level events 488
Event listeners 469, 470
  with lambda expressions 474
Event objects 469
Event procedures 469
Event sources 469, 470
EventListenerList class 641
EventObject class 469
  getActionCommand, getSource methods 487
Evins, Jim 589
Exceptions
  FileNotFoundException 243
  IllegalStateException 27, 690
  in SQL 279
  IndexOutOfBoundsException 690
  InvalidObjectException 131
  InvalidPathException 87
  IOException 62, 214
  MissingResourceException 370
  NotSerializableException 125
  OverlappingFileLockException 111
  ParseException 347
  ReadOnlyBufferException 102
  SocketTimeoutException 215
  SQLException 279, 295, 311
  SQLWarning 295
  SyncProviderException 302
  UnknownHostException 214
Exclusive lock 111
exclusiveOr method 664
  of Area 664
exec method
  of Runtime 132
execute method
  of CachedRowSet 301, 302
  of RowSet 302
  of Statement 278, 282, 293, 294
executeBatch method 312
  of Statement 313
executeLargeBatch method
  of Statement 313
executeLargeUpdate method
  of Statement 277
executeQuery method
  of PreparedStatement 285, 289
  of Statement 276, 277, 295, 297
executeUpdate method
  of PreparedStatement 285, 289
  of Statement 276, 277, 294, 310, 311
executor method
  of HttpClient.Builder 253
ExecutorService interface 380
exists method
  of Files 93, 94
EXIT 273
exportNode method
  of Preferences 495
exportSubtree method
  of Preferences 495
export*Xxx* methods
  of Preferences 492
Extension class loader 399
extern "C", in native methods (C++) 744
External entities 167
External padding 549
Externalizable interface
  methods of 127, 128

F

F (float), type code 119
Factoring algorithms 423
fatalError method 169
  of ErrorHandler 169
Field class
  getName, getType methods 642
Fields
  of deserialized objects 128
  transient 125
File class 89
  separator constant 64
  toPath method 88, 89
File dialogs 572
  adding accessory components to 576
File pointers 80
File systems, POSIX-compliant 94
file.encoding package 76
file: scheme 231
FileChannel class 106, 112
  lock method 110, 112, 113
  map method 102, 106
  open method 102, 106
  truncate method 113
  tryLock method 110, 113
FileFilter interface 574
FileFilter class 578
  accept method 578
  getDescription method 578
FileFilter class (Swing), methods of 574
FileInputStream class 64, 66, 105
  Constructor 66
  getChannel method 105
  read method 58
fileKey method
  of BasicFileAttributes 95
FileLock class 113
  close method 113
  isShared method 110
FileNameExtensionFilter class 578
  Constructor 578
FileNotFoundException class 243
FileOutputStream class 64, 66, 106
  Constructor 66
  getChannel method 106
Files class 8, 86, 89, 90, 91, 93, 94, 100
  channels for 102
  closing 95, 98
  configuration 110
  copy method 91, 93, 100
  copying 91
  createDirectories method 91
  createDirectory method 91
  createFile method 91
  createLink method 91
  createSymbolicLink method 91
  createTempDirectory method 91
  createTempFile method 91

createXxx methods 90
creating 90
delete method 93
delete, deleteIfExists methods 92
deleteIfExists method 93
deleting 92
encrypted data in 437
exists method 93, 94
filtering 97, 98, 689
filters for 574, 576
find method 96
getOwner method 94
hierarchical structure of 613
isDirectory method 94
isExecutable method 94
isHidden method 94
isReadable method 94
isRegularFile method 94
isSymbolicLink method 94
isWritable method 94
isXxx methods 93
lines method 5, 8, 53, 89
list method 95
locking 110
memory-mapped 53, 101
MIME type of 89
mismatch method 89, 90
missing 381
move method 91, 93
moving 91
newBufferedReader method 90
newBufferedWriter method 90
newBufferedXxx methods 90
newDirectoryStream method 98, 100
newInputStream method 90
newOutputStream method 90
newXxxStream methods 90
opening/saving in GUI 572
probeContentType method 89, 90
random-access 80, 102
readAllBytes method 90
readAllLines method 90
readAllXxx methods 89
readAttributes method 94
reading/writing 64, 89
by one byte 58
readString method 89, 90
size method 93, 94
total number of bytes in 81
traversing 98
walk method 95, 96
walkFileTree method 98, 100
with multiple images 690
write method 90
write, writeString methods 89
writeString method 90
FileSystem class 97, 101
getPath method 100, 101
getPathMatcher method 97
FileSystems class 98, 101
close method 101
getDefault method 98
newFileSystem method 100, 101
FileTime class 95
toInstant method 95, 337
FileTypeDetector 90
FileView class 578
getDescription method 579
getIcon method 579
getName method 578
getTypeDescription method 579
isTraversable method 579
FileView class, methods of 575
FileVisitor interface 99
methods of 98
Fill 649, 664
fill field
of GridBagConstraints 555
fill method
of Graphics2D 460, 461, 650
Filling shapes 649
fillXxx methods
of Graphics 650
filter method
of BufferedImageOp 703, 709
of Optional 16
of Stream 2, 3, 9, 10, 13
FilteredRowSet interface 300
filtering method
of Collectors 32, 35
Filters
for images 703
for numbers 594
for table rows 594, 595
glob patterns for 97, 98
implementing 595
FilterXxxStream classes 65
Final block padding 432
find method
of Files 96
of Matcher 147
of SymbolLookup 759
findAll method
of Scanner 148
findAny method 14
of Stream 14
findClass method 401
of ClassLoader 405
findColumn method
of ResultSet 278
findFirst method 13
of Stream 14
findOrThrow method
of SymbolLookup 759
findStatic method
of MethodHandle.Lookup 762
Fingerprints. See Digital fingerprints
finisher method
of Collector 38
first method
of ResultSet 296, 299
firstDayOfMonth method
of TemporalAdjusters 326
firstDayOfNextMonth method
of TemporalAdjusters 326
firstDayOfNextYear method
of TemporalAdjusters 326
firstDayOfYear method

of TemporalAdjusters 326
firstValue method 249
of HttpHeaders 254
firstVariadicArg method
of Linker.Option 765
#FIXED 167
Flash 444
flatMap method
of Optional 18, 19, 21
of Stream 10, 11
flatMapping method 32
of Collectors 35
flip method
of Buffer 108, 109
FLOAT 271, 313
float type
printing 70
streams of 46
type code for 119
writing in binary format 78
FloatBuffer class 107
Floating-point numbers 340, 346
Flow layout manager 501
FlowLayout class 503
Constructor 503
flush method
of CipherOutputStream 438
of Closeable 62
of Flushable 62, 63
of OutputStream 58, 60
Flushable interface 62, 63
close method 62
flush method 62, 63
fn (SQL escape) 292
FocusEvent class 488
isTemporary method 489
FocusListener interface, methods of 489
fold method
of Gatherers 45
Folders, icons for 621, 631
followRedirects method 247
of HttpClient.Builder 253
Font class 462, 466
Constructor 466
getFamily method 466
getFontName method 466
getLineMetrics method 464, 466
getName method 466
getStringBounds method 463, 466
Font render context 679
FontMetrics class 467
getFontRenderContext method 467
Fonts 461
checking availability of 462
names of 461, 462
size of 462
styles of 462
typesetting properties of 463
forEach method 22
of Stream 22
forEachOrdered method 22
Foreground color 461
Foreign function
callback 759
Foreign functions 747
Foreign Functions and Memory API 741
Forest 614, 620
ForkJoinPool class
commonPool method 53
forLanguageTag method 342
of Locale 345
Format class 336, 363
format method 363
format method
of DateTimeFormatter 331, 335, 356
of Format 363
of LocalDate 353
of LocalDateTime 353, 354
of LocalTime 353, 354
of MessageFormat 362, 363
of NumberFormat 347, 349
of String 345
of ZonedDateTime 353
Formatting
dates 340, 353, 361
messages 361
numbers 340, 346, 361
Forms, processing 239
forName method
of Charset 75
of Class 383
ForwardingJavaFileManager class 387
ForwardingJavaFileManager method 387
getJavaFileForOutput method 387
ForwardingJavaFileManager method
of ForwardingJavaFileManager 387
Frame class 445, 449
getIconImage method 449
getTitle method 449
isResizable method 449
setIconImage method 447, 449
setResizable method 447, 449
setTitle method 447, 449
Frames
closing by user 446
creating 445
displaying information in 450
positioning 447
properties of 447
FROM 269
from method
of Instant 336
of ZonedDateTime 336
FTP 235
ftp: scheme 231, 235
Function interface
identity method 27
FunctionDescriptor interface 759
of method 759
ofVoid method 759
Functions. See Methods

G

Gadget chains 132
Garbage collection
native methods and 745
gather method

of Stream 45
Gatherer interface 46
initializer method 46
of method 46
ofSequential method 46
Gatherer.Downstream interface 45
isRejecting method 46
push method 45
Gatherer.Integrator interface 46
integrate method 46
of method 46
Gatherers class 45
fold method 45
mapConcurrent method 45
windowFixed method 45
GeneralPath class 650, 655, 663
Constructor 663
Generate 47
generate method
of Stream 4, 7
generateKey method 433
of KeyGenerator 437
Generators, converting to streams 52
Generic programming
classes in 522
GET 239, 241
building 247
get method
of AttributeSet 734, 738
of Bindings 391
of ByteBuffer 103, 106
of CharBuffer 107
of HttpRequest.Builder 247, 254
of MemorySegment 751
of Optional 17, 18, 20
of Paths 87
of Preferences 491, 494
of ScriptEngine 390
of ScriptEngineManager 390
of Supplier 9
getActionCommand method
of ActionEvent 488
of ButtonModel 517, 519
of EventObject 487
getActionMap method
of JComponent 482
getAddress method 217
of InetAddress 218
getAdjustable, getAdjustmentType methods (AdjustmentEvent) 488
getAdvance method
of TextLayout 680
getAllByName method 217
of InetAddress 218
getAllowsChildren method
of TreeNode 622
getAncestorOfClass method 568
of SwingUtilities 571
getAscent method
of LineMetrics 466
of TextLayout 680
getAs*Xxx* method
of OptionalDouble 51
of OptionalInt 51
of OptionalLong 51
getAs*Xxx* methods
of Optional*Xxx* 48
getAtIndex method
of MemorySegment 751
getAttribute method 158
of Element 161
getAttributeCount method
of XMLStreamReader 192
getAttributeLocalName method
of XMLStreamReader 192
getAttributeName method
of XMLStreamReader 192
getAttributes method
of DocPrintJob 738
of Node 158, 161
of PrintService 738
getAttributeValue method
of XMLStreamReader 192
getAttribute*Xxx* methods
of XMLStreamReader 190
getAuthority method 232
getAutoCommit method 312
of Connection 312
getAutoCreateRowSorter method
of JTable 585
getAvailableCurrencies method
of Currency 353
getAvailableFontFamilyNames method
of GraphicsEnvironment 462
getAvailableLocales method
of Collator 361
of NumberFormat 343, 347, 348
getAvailableZoneIds method
of ZoneId 328
getAverage method 23
of DoubleSummaryStatistics 25, 51
of IntSummaryStatistics 25, 51
of LongSummaryStatistics 25, 51
getBackground method
of Component 461
getBinaryStream method 290
of Blob 291
getBlob method
of ResultSet 290, 291
getBlockSize method
of Cipher 436
getBoolean method
of Preferences 491, 494
getBundle method 370
of ResourceBundle 373
getByName method 216
of InetAddress 218
getByteArray method 491
of Preferences 494
getBytes method 290
of Blob 291
getCandidateLocales method
of ResourceBundle.Control 370
getCategory method 733
of Attribute 738
getCellEditorValue method 605, 607, 608
of CellEditor 613
getCellRenderer method 604

getCellSelectionEnabled method
of JTable 600
getCenterX method
of RectangularShape 459
getCenterX/Y methods
of RectangularShape 456
getCenterY method
of RectangularShape 459
getChannel method
of FileInputStream 105
of FileOutputStream 106
of RandomAccessFile 106, 113
getChar method 103
getCharacterStream method 290
of Clob 291
getCharContent method
of SimpleJavaFileObject 387
getChild method 640, 642
of TreeModel 646
getChildAt method
of TreeNode 629
getChildCount method
of TreeModel 640, 642, 646
of TreeNode 629
getChildNodes method 156
of Node 161
getClassLoader method 398
of Class 404
getClickCount method 482, 489
of MouseEvent 486
getClip method 713
of Graphics 680
getClob method
of ResultSet 291
getClob method (ResultSet) 290
getCollationKey method 358
of Collator 361
getColorModel method 698
of BufferedImage 701
getColumn method 590
of TableColumnModel 600
getColumnClass method 588
of TableModel 599
getColumnCount method
of ResultSetMetaData 304, 309
of TableModel 585, 586, 588
getColumnDisplaySize method
of ResultSetMetaData 309
getColumnLabel method
of ResultSetMetaData 309
getColumnModel method 590
of JTable 599
getColumnName method 586
of ResultSetMetaData 309
of TableModel 588
getColumnNumber method
of Diagnostic 386
of SAXParseException 170
getColumns method
of JTextField 508
getColumnSelectionAllowed method
of JTable 600
getColumnType method 605
getColumn*Xxx* methods
of ResultSetMetaData 304
getCommand method
of RowSet 302
getCompactNumberInstance method
of NumberFormat 349
getComponentPopupMenu method
of JComponent 536
getConcurrency method 295, 297
of ResultSet 299
getConnection method 273, 282, 315
of DriverManager 275
getConnectTimeout method
of URLConnection 238
getContent method
of URLConnection 239
getContentEncoding method
of URLConnection 239
getContentEncoding, getContentType methods
of URLConnection 234, 243
getContentEncoding, getContentType methods (URLConnection)
236
getContentLength method 234, 236
of URLConnection 239
getContentType method
of URLConnection 239
getContext method
of ScriptEngine 391
getContextClassLoader method 400
of Thread 405
getCount method
of DoubleSummaryStatistics 25, 51
of IntSummaryStatistics 25, 51
of LongSummaryStatistics 25, 51
getCountry method 30
of Locale 345
getCrc method
of ZipEntry 86
getCurrencyCode method
of Currency 353
getCurrencyInstance method 346, 352
of NumberFormat 348
getData method 157
of CharacterData 161
getDataElements method
of ColorModel 702
of Raster 698, 699, 701
getDataSize method
of DataTruncation 281
getDate method
of ResultSet 277
of URLConnection 234, 236, 239
getDayOfMonth method
of LocalDate 324
of ZonedDateTime 330
getDayOfWeek method
of LocalDate 324
of ZonedDateTime 330
getDayOf*Xxx* methods
of LocalDate 322
getDayOfYear method
of LocalDate 324
of ZonedDateTime 330
getDecompositon method
of Collator 361

getDefault method 343
of FileSystems 98
of Locale 345
of SSLServerSocketFactory 230
of SSLSocketFactory 230
getDefaultEditor method
of JTable 612
getDefaultFractionDigits method
of Currency 353
getDefaultName method
of NameCallback 417
getDefaultRenderer method 604
of JTable 612
getDefaultToolkit method 448
of Toolkit 449
getDescent method
of LineMetrics 466
of TextLayout 680
getDescription method
of FileFilter 574, 578
of FileView 575, 579
getDiagnostics method
of DiagnosticCollector 386
getDisplayCountry method
of Locale 345
getDisplayCountry, getDisplayLanguage methods
of Locale 27
getDisplayLanguage method
of Locale 345
getDisplayName method
of DayOfWeek, Month 332, 354
of Locale 344, 345, 347
getDocumentElement method 155
of Document 161
getDoInput method
of URLConnection 238
getDoOutput method
of URLConnection 238
getDouble method
of ByteBuffer 103
of Preferences 491, 494
of ResultSet 277
getEngineByExtension method
of ScriptEngineManager 388
getEngineByMimeType method
of ScriptEngineManager 388
getEngineByName method
of ScriptEngineManager 388
getEngineFactories method
of ScriptEngineManager 388
getEngine*Xxx* methods
of ScriptEngineManager 388
getErrorCode method 279
of SQLException 281
getErrorStream method 243
of HttpURLConnection 246
getErrorWriter method
of ScriptContext 391
getExpiration method 234, 236
of URLConnection 239
getExtensions method
of ScriptEngineFactory 389
getFamily method
of Font 466
getFields method 642
getFileName method 88
of Path 89
getFilePointer method 80
of RandomAccessFile 84
getFileSuffixes method
of ImageReaderWriterSpi 696
getFillsViewportHeight method
of JTable 585
getFirstChild method 157
of Node 161
getFloat method 103
of Preferences 491, 494
getFont method
of Component 509
getFontMetrics method 464
of JComponent 466
getFontName method
of Font 466
getFontRenderContext method 679
of FontMetrics 467
of Graphics2D 463, 466, 680
getForeground method
of Component 461
getFormatNames method
of ImageReaderWriterSpi 696
getFragment method 232
getHeaderField method
of URLConnection 238
getHeaderFieldKey method
of URLConnection 238
getHeaderFields method
of URLConnection 238
getHeader*Xxx* methods
of URLConnection 234, 236
getHeight method
of ImageReader 691, 695
of LineMetrics 466
of PageFormat 713, 718
of RectangularShape 456, 459
getHost method 232
getHostAddress method
of InetAddress 218
getHostName method
of InetAddress 218
getHour method
of LocalTime 327
of ZonedDateTime 331
getIcon method
of FileView 575, 579
of JLabel 510
getIconImage method
of Frame 449
getIdentifier method
of RowFilter.Entry 602
getIfModifiedSince method
of URLConnection 238
getImage method
of ImageIcon 450, 467
getImageableHeight method
of PageFormat 718
getImageableWidth method
of PageFormat 718
getImageableX method

of PageFormat 718
getImageable*Xxx* methods
of PageFormat 713
getImageableY method
of PageFormat 718
getImageReadersByFormatName method
of ImageIO 695
getImageReadersByMIMEType method
of ImageIO 695
getImageReadersBySuffix method
of ImageIO 695
getImageWritersByFormatName method
of ImageIO 695
getImageWritersByMIMEType method
of ImageIO 695
getImageWritersBySuffix method
of ImageIO 695
getImage*Xxx*By*Xxx* methods
of ImageIO 689
getIndex method
of DataTruncation 281
getIndexOfChild method 640
of TreeModel 646
getInheritsPopupMenu method
of JComponent 536
getInputMap method 480
of JComponent 482
getInputStream method
of Socket 215, 218
of URLConnection 234, 239, 242, 243
getInstance method
of AlphaComposite 683, 688
of Cipher 431, 436
of Collator 361
of Currency 352, 353
of KeyGenerator 436
of Locale 357
of MessageDigest 420, 421
getInt method
of ByteBuffer 103
of Preferences 491, 494
of ResultSet 277
getInterface method
of Invocable 393
getISOCountries method 343
of Locale 345
getISOLanguages method 343
getItem, getItemSelectable methods
of ItemEvent 488
getItemAt method 522
getJavaFileForOutput method
of ForwardingJavaFileManager 387
getJavaFileObjectsFromFiles method
of StandardJavaFileManager 386
getJavaFileObjectsFromPaths method
of StandardJavaFileManager 386
getJavaFileObjectsFromStrings method
of StandardJavaFileManager 386
getJDBCMajorVersion method
of DatabaseMetaData 308
getJDBCMinorVersion method
of DatabaseMetaData 308
getKeys method 372
of ResourceBundle 373
getKeyStroke method 479
of KeyStroke 481
getKey*Xxx* methods
of KeyEvent 489
getKind method
of Diagnostic 386
getLanguage method
of Locale 345
getLargeUpdateCount method
of Statement 278
getLastChild method 158
of Node 161
getLastModifed method
of URLConnection 239
getLastModified method 234, 236
getLastPathComponent method 623
of TreePath 629
getLastSelectedPathComponent method 624
of JTree 628
getLeading method
of LineMetrics 466
of TextLayout 680
getLength method
of Attributes 189
of NamedNodeMap 162
of NodeList 156, 162
getLineMetrics method 464
of Font 466
getLineNumber method
of Diagnostic 386
of SAXParseException 170
getLocale method
of MessageFormat 363
getLocalHost method 217
of InetAddress 218
getLocalName method
of Attributes 189
of Node 184
of XMLStreamReader 192
getLong method 103
of Preferences 491, 494
getMax method 23
of DoubleSummaryStatistics 25, 51
of IntSummaryStatistics 25, 51
of LongSummaryStatistics 25, 51
getMaxConnections method
of DatabaseMetaData 309
getMaximumFractionDigits method
of NumberFormat 349
getMaximumIntegerDigits method
of NumberFormat 349
getMaxStatements method 279
of DatabaseMetaData 309
getMaxX method
of RectangularShape 459
getMaxY method
of RectangularShape 459
getMessage method
of Diagnostic 386
getMetaData method
of Connection 303, 308
of PreparedStatement 309
of ResultSet 304, 309
getMethodCallSyntax method 392

getMimeTypes method
of ImageReaderWriterSpi 696
of ScriptEngineFactory 389
getMin method
of DoubleSummaryStatistics 25, 51
of IntSummaryStatistics 25, 51
of LongSummaryStatistics 25, 51
getMinimumFractionDigits method
of NumberFormat 349
getMinimumIntegerDigits method
of NumberFormat 349
getMinute method
of LocalTime 327
of ZonedDateTime 331
getMinX method
of RectangularShape 459
getMinY method
of RectangularShape 459
getModel method
of RowFilter.Entry 602
getModifiers method
of ActionEvent 488
getMonth method
of LocalDate 324
of ZonedDateTime 331
getMonthValue method
of LocalDate 324
of ZonedDateTime 331
getMoreResults method 293
of Statement 294
getName method
of Attribute 738
of Field 642
of FileView 575, 578
of Font 466
of NameCallback 417
of Principal 412
of PrintService 728
of XMLStreamReader 192
of ZipEntry 86
getNames method
of ScriptEngineFactory 389
getNamespaceURI method 184
of Node 184
getNano method
of LocalTime 327
of ZonedDateTime 331
getNewState, getOldState methods
of WindowEvent 490
getNextEntry method 84
of ZipInputStream 85
getNextException method 279
of SQLException 280
getNextSibling method 158
of Node 161
getNextWarning method
of SQLWarning 281
getNodeName method
of Node 161
getNodeValue method
of Node 161
getNode*Xxx* methods (Node) 158, 184
getNumberInstance method 346
of NumberFormat 348
getNumericCode method
of Currency 353
getNumericCodeAsString method
of Currency 353
getNumImages method
of ImageReader 695
getNumThumbnails method
of ImageReader 695
getNum*Xxx* methods
of ImageReader 690
getObject method
of ResourceBundle 371, 372, 373
of ResultSet 277, 278
getOffset method
of ZonedDateTime 331
getOppositeWindow method 489
getOrientation method
of PageFormat 718
getOriginatingProvider method
of ImageReader 689, 695
of ImageWriter 696
getOutputSize method
of Cipher 436
getOutputStream method
of Socket 215, 218
of URLConnection 234, 239, 241
getOwner method 94
getPageCount method 719
getPageSize method
of CachedRowSet 303
getPaint method
of Graphics2D 461
getParameter method
of DataTruncation 281
getParameterClassName method
of ParameterMetaData 309
getParameterCount method
of ParameterMetaData 309
getParameterMode method
of ParameterMetaData 309
getParameterType method
of ParameterMetaData 309
getParameterTypeName method
of ParameterMetaData 309
getParent method
of ClassLoader 405
of Path 88, 89
of TreeNode 629, 630
getParentNode method
of Node 161
getPassword method
of JPasswordField 510
of PasswordCallback 418
of RowSet 302
getPath method
of FileSystem 100, 101
of TreeSelectionEvent 639
of URI 232
getPathMatcher method
of FileSystem 97
getPaths method 635
of TreeSelectionEvent 639
getPathToRoot method 625
getPercentInstance method 346

of NumberFormat 348
getPixel method
of Raster 702
getPixel, getPixels methods
of Raster 698
getPixels method
of Raster 702
getPoint method 489
of MouseEvent 486
getPointCount method 656
getPort method 232
getPrecision method
of ParameterMetaData 310
getPredefinedCursor method 483
getPreferredSize method 452
of Component 453
getPreviousSibling method
of Node 161
getPrincipals method
of Subject 412
getPrinterJob method 711
of PrinterJob 717
getPrintService method 729
of StreamPrintServiceFactory 729
getPrompt method
of NameCallback 417
of PasswordCallback 418
getQName method
of Attributes 189
getQuery method 232
getRaster method 697
of BufferedImage 701
getReader method
of ScriptContext 391
getReaderFileSuffixes method
of ImageIO 695
getReaderFormatNames method
of ImageIO 695
getReaderMIMETypes method
of ImageIO 695
getReader*Xxx* methods
of ImageIO 689
getReadTimeout method
of URLConnection 238
getRequestProperties method
of URLConnection 238
getResponseCode method 244
getResultSet method
of Statement 278
getRGB method
of Color 702
of ColorModel 699, 702
getRoot method
of Path 88, 89
of TreeModel 640, 642, 646
getRootPane method 568
of JComponent 572
getRotateInstance method 676
of AffineTransform 677
getRow method 296
of ResultSet 299
getRowCount method 585, 586
of TableModel 588
getRowHeight method
of JTable 600
getRowMargin method
of JTable 600
getRowSelectionAllowed method
of JTable 600
getSavepointId method
of Savepoint 312
getSavepointName method
of Savepoint 312
getScale method
of ParameterMetaData 310
getScaleInstance method 676
of AffineTransform 677
getScreenSize method 448
of Toolkit 449
getScrollAmount method 489
getSecond method
of LocalTime 327
of ZonedDateTime 331
getSeconds method 318
getSelectedColumns method 592, 596
getSelectedFile method
of JFileChooser 578
getSelectedFile/Files methods
of JFileChooser 574
getSelectedFiles method
of JFileChooser 578
getSelectedItem method 522
of JComboBox 524
getSelectedObjects method 517
getSelectedRows method 592
getSelection method 517
of ButtonGroup 519
getSelectionModel method
of JTable 600
getSelectionPath method 623
of JTree 628, 639
getSelectionPaths method 635
of JTree 639
getShearInstance method 676
of AffineTransform 677
getShort method 103
getSize method
of Component 449
of ZipEntry 86
getSource method 487
of Diagnostic 386
getSQLState method 279
of SQLException 281
getSQLStateType method 279
getStandardFileManager method
of JavaCompiler 386
getStateChange method 488
getStrength method
of Collator 361
getString method
of MemorySegment 750
of ResourceBundle 371, 373
of ResultSet 277
getStringArray method
of ResourceBundle 373
getStringBounds method 463
of Font 466
GetStringUTFChars 745

getStringValue method
of RowFilter.Entry 602
getSubject method
of LoginContext 411
getSubString method 290
of Clob 291
getSum method
of DoubleSummaryStatistics 25, 51
of IntSummaryStatistics 25, 51
of LongSummaryStatistics 25, 51
getSymbol method
of Currency 353
getSystemClassLoader method
of ClassLoader 405
getSystemJavaCompiler method 379
getTableCellEditorComponent method 607, 608
of TableCellEditor 612
getTableCellRendererComponent method 602
of TableCellRenderer 612
getTableHeader method 584
getTableName method
of CachedRowSet 303
getTables method 304
of DatabaseMetaData 308
getTagName method 156, 184
of Element 161
getTask method 380
of JavaCompiler 386
getText method
of JLabel 510
of JTextComponent 507, 508
of XMLStreamReader 192
getTimeZone method 337
getTitle method
of Frame 449
getTransferSize method
of DataTruncation 281
getTranslateInstance method 676, 679
of AffineTransform 677
getTreeCellRendererComponent method
of DefaultTreeCellRenderer 632
of TreeCellRenderer 632, 633
getType method
of Field 642
of ResultSet 295, 299
getTypeDescription method 575
of FileView 579
getUpdateCount method 293
of Statement 278
getURI method
of Attributes 189
getURL method
of RowSet 302
getURLs method 399
getUserInfo method 232
getUsername method
of RowSet 302
getValue method
of Action 477, 481
of AdjustmentEvent 488
of Attributes 189
of Copies 734
of RowFilter.Entry 602
getValueAt method 585, 586, 605
of TableModel 588
getValueCount method
of RowFilter.Entry 602
getVendorName method
of IIOServiceProvider 696
getVendorName, getVersion methods
of IIOServiceProvider 689
getVersion method
of IIOServiceProvider 696
getWarnings method
of Connection 281
getWheelRotation method 489
getWidth method
of ImageReader 691, 695
of PageFormat 713, 718
of Rectangle2D 455
of RectangularShape 456, 459
getWindow method 489
getWriter method
of ScriptContext 391
getWriterFileSuffixes method
of ImageIO 695
getWriterFormatNames method
of ImageIO 695
getWriterMIMETypes method
of ImageIO 695
getWriter*Xxx* methods
of ImageIO 689
getX method
of MouseEvent 486
of RectangularShape 459
getX/Y methods
of MouseEvent 482, 489
get*Xxx* method
of ByteBuffer 107
of Period 325
of ResultSet 278
getY method
of MouseEvent 486
of RectangularShape 459
getYear method
of LocalDate 324
of ZonedDateTime 331
GIF 688
animated 690
image manipulations on 706
printing 726
GlassFish server 315
Glob patterns 97, 98
Global arena 748
GlobalSign 427
GMail 259, 260
Gnu C compiler 745
Google Maps 240
GradientPaint class 671, 672, 673
constructor 671, 673
cyclic parameter 672
Grapheme clusters 9
Graphic Java™ (Geary) 581, 614
Graphical User Interface 443
components of 497
choice components 513
dialog boxes 558
menus 530

text input 507
toolbars 543
tooltips 545
events in 469
keyboard focus in 479
layout of 501, 546
Graphics class 454, 467, 469, 647, 680
copyArea method 469
drawImage method 467, 469
drawXxx, fillXxx methods 650
get/setClip methods 678, 713
getClip method 680
setClip method 680
Graphics editor applications 482
Graphics2D class 454, 461, 466, 647, 649, 671, 673, 678, 680, 687
clip method 648, 678, 680, 713
draw method 454, 649, 650, 664
drawString method 466
fill method 460, 461, 649, 650, 664
getFontRenderContext method 463, 466, 679, 680
getPaint method 461
rotate method 674, 675, 678
scale method 673, 674, 678
setComposite method 648, 682, 687
setPaint method 460, 461, 648, 671, 673
setRenderingHint, setRenderingHints methods 648
setStroke method 648, 664, 671
setTransform method 676, 678
shear method 674, 678
transform method 648, 676, 678
translate method 674, 678, 720
GraphicsEnvironment class 462
Greenwich Royal Observatory 318, 328
GregorianCalendar class
toZonedDateTime method 336
Grid bag layout 546
Grid layout 506
GridBagConstraints class 548, 554
anchor field 554
anchor, fill parameters 549
Constructor 555
fill field 555
gridwidth field 554
gridx field 554
gridx/y, gridwidth/height parameters 548
helper class for 550
insets field 555
insets parameter 549
ipadx field 555
weightx field 554
weightx/y parameters 548
GridLayout class 503, 506
Constructor 506
gridwidth field
of GridBagConstraints 554
gridx field
of GridBagConstraints 554
Groovy 387, 388
Group layout 546
group method
of MatchResult 147
groupElement method
of MemoryLayout.PathElement 753
Grouping 30
classifier functions of 30
reducing to numbers 31
groupingBy method 30
of Collectors 30, 34
groupingByConcurrent method 30, 52
of Collectors 30
GUI. See Graphical User Interface
Gödel's theorem 406

H

Half-closing connections 223
Handbook of Applied Cryptography, The (Menezes et al.) 422
handle method
of CallbackHandler 417
handleGetObject method 372
of ResourceBundle 373
Handles 617, 631
HashSet class
readObject, writeObject methods 126
HashXxxAttributeSet classes 711, 732
Haskell 387
hasMatch method
of Matcher 147
hasNext method
of XMLStreamReader 192
hasRemaining method
of Buffer 106
Header information, from server 234
header method
of HttpRequest.Builder 254
Headers 249
Headers (Swing tables)
rendering 604
scrolling 582
headers method
of HttpResponse 254
Height 463
Helper classes 550
Hex editors
creating class files in 406
modifying bytecodes with 408
Hidden commands, in XML comments 154
Hosts 216
HTML
attributes in 151, 153
end and empty tags in 151
forms in 239
generating from XML files 201, 204
in labels 509
mixing with JSP 383
printing 726
tables in 547
vs. XML 151
HTML editors 499
HTTP 265
redirects between HTTPS and 244
request headers in 235
http: scheme 231
HttpClient class 247, 253
enabling logging for 249
newBuilder method 247, 249, 253

newHttpClient method 247, 253
send method 253
sendAsync method 249, 253
HttpClient.Builder interface 253
build method 247, 248, 253
executor method 253
followRedirects method 247, 253
HttpHeaders class 254
firstValue method 249, 254
map method 249, 254
HttpRequest class 254
newBuilder method 247, 249, 254
HttpRequest.Builder interface 254
build method 254
DELETE method 254
GET method 247, 254
header method 254
POST method 248, 254
PUT method 254
uri method 247, 249, 254
HttpResponse class
methods of 248
HttpResponse interface 254
body method 254
headers method 254
statusCode method 254
HttpResponse.BodyHandlers class
discarding method 249
ofString method 248, 249
HTTPS 244
https: scheme 231
HttpURLConnection class 244, 246
getErrorStream method 243, 246
getResponseCode method 244
setInstanceFollowRedirects method 244

I

I (int), type code 119
IANA 328
IBM 149
DB2 database 271
DOM parser 154
IBM437 encoding 367
ICC profiles 698
Icons
in column headers 604
in menu items 533
in sliders 527
in table cells 602
in trees 621, 631
ID, IDREF, IDREFS attribute types (DTDs) 166, 167
identity method
of Function 27
Identity values 39
IDs, uniqueness of 167, 174
IETF BCP 47 341
IFC 444
ifPresent method
of Optional 15, 16
of OptionalDouble 51
of OptionalInt 51
of OptionalLong 51
ifPresentOrElse method 15
of Optional 16
IIOImage class 691, 696
Constructor 696
IIOServiceProvider class 696
getVendorName method 696
getVersion method 696
get*Xxx* methods 689
IllegalAccessException class 642
IllegalStateException class 27, 690
ImageIcon class 449, 450
Constructor 450
getImage method 450, 467
ImageInputStream interface 690
"Seek forward only" mode 690
ImageIO class 694
createImageInputStream method 695
createImageOutputStream method 695
createImage*Xxx*Stream methods 690
determining image type 688, 689
getImageReadersByFormatName method 695
getImageReadersByMIMEType method 695
getImageReadersBySuffix method 695
getImageWritersByFormatName method 695
getImageWritersByMIMEType method 695
getImageWritersBySuffix method 695
getImage*Xxx*By*Xxx* methods 689
getReaderFileSuffixes method 695
getReaderFormatNames method 695
getReaderMIMETypes method 695
getReader*Xxx*, getWriter*Xxx* methods 689
getWriterFileSuffixes method 695
getWriterFormatNames method 695
getWriterMIMETypes method 695
read method 694
read, write methods 688
write method 695
ImageOutputStream interface 691
ImageReader class 688, 689, 695
getHeight method 691, 695
getNumImages method 695
getNumThumbnails method 695
getNum*Xxx* methods 690
getOriginatingProvider method 689, 695
getWidth method 691, 695
read method 695
readThumbnail method 695
setInput method 695
ImageReaderWriterSpi class 696
getFileSuffixes method 696
getFormatNames method 696
getMIMETypes method 696
Images
blurring 705
color values of 698
edge detection of 705
filtering 703
getting size of, before reading 691
incremental rendering of 696
manipulating 696
metadata in 691
multiple, in a file 690
printing 710, 726, 729
raster 688
constructing from pixels 697

readers/writers for 688
rotating 703, 704
superimposing 680, 681
thumbnails for 690
vector 647
Images, displaying 467
ImageWriter class 688, 691, 696
canInsertImage method 691, 696
getOriginatingProvider method 696
setOutput method 696
write method 696
write, writeInsert methods 691
writeInsert method 696
#IMPLIED 167
import keyword 401
importPreferences method 492
of Preferences 495
include method
of RowFilter 595, 601
Incremental rendering 696
Indexed color model 704
IndexOutOfBoundsException class 690
InetAddress class 218
getAddress method 218
getAllByName method 218
getByName method 218
getHostAddress method 218
getHostName method 218
getLocalHost method 218
get*Xxx* methods 216
InetSocketAddress class 227
Constructor 227
isUnresolved method 227
Inheritance trees 280
init method
of Cipher 436
of KeyGenerator 437
initialize method
of LoginModule 418
initializer method
of Gatherer 46
Input dialogs 561
Input maps 480, 481
Input streams 57
as input source 155
buffered 65
byte processing in 65
byte-oriented 57
chaining 65
closing 58
filters for 64
hierarchy of 60
keeping open 223
objects in 113
redirecting 391
Unicode 57
InputSource class 169
Constructor 169
InputStream class 57, 59, 62
available method 58, 59
close method 58, 59
mark method 59
markSupported method 59
nullInputStream method 59
read method 57, 59
readAllBytes method 59
readAllBytes, readNBytes methods 58
readNBytes method 59
reset method 59
skip method 59
skipNBytes method 59
transferTo method 59
InputStreamReader class 67
INSERT 270
executing 276
in batch updates 311
vs. methods of ResultSet 298
insert method
of JMenu 532
insertItemAt method 523
of JComboBox 524
insertNodeInto method 624
of DefaultTreeModel 629
insertRow method 297
of ResultSet 299
insertSeparator method
of JMenu 532
insets field
of GridBagConstraints 555
INSTANCE 128
Instant class 318, 320
compareTo method 318
equals method 318
from method 336
immutability of 319
legacy classes and 336
minus method 320
minus*Xxx* method 320
now method 318, 320
plus method 320
plus*Xxx* method 320
int type
printing 70
storing 78
streams of 46
type code for 119
writing in binary format 78
IntBuffer class 107
INTEGER 271, 313
IntegerSyntax class 734
integrate method
of Gatherer.Integrator 46
Intel processor, little-endian order in 78
IntelliJ IDE 125
Interfaces
accessing script classes with 393
implementing in script engines 392
methods in
do-nothing 475
Intermittent bugs 446
Internal padding 549
Internationalization 339, 377
Internet Engineering Task Force 341
Interpolation 703
for gradients 671
strategies of 703
when transforming images 703
Interruptible sockets 224

intersect method
of Area 664
ints method
of RandomGenerator 47, 50
IntStream interface 46, 49
average method 48, 49
boxed method 47, 49
mapToInt method 47
max method 49
max, min methods 48
min method 49
of method 47, 49
range method 49
range, rangeClosed methods 47
rangeClosed method 49
sum method 49
sum, summaryStatistics methods 48
summaryStatistics method 49
toArray method 47, 49
IntSummaryStatistics class 23, 25, 48, 51
getAverage method 25, 51
getCount method 25, 51
getMax method 25, 51
getMin method 25, 51
getSum method 25, 51
intValue method 347
Invalid pointers (C, C++) 742
InvalidObjectException class 131
InvalidPathException class 87
Invocable interface 392, 393
getInterface method 393
invokeFunction method 393
invokeMethod method 393
invoke*Xxx* methods 392
invoke method
of MethodHandle 759
invokeFunction method
of Invocable 393
invokeMethod method
of Invocable 393
IOException class 62, 68, 214
IP addresses 212, 216
ipadx field
of GridBagConstraints 555
IPP (Internet Printing Protocol) 1.1 738
IPv6 addresses 216
isActionKey method 489
isAfter method
of LocalDate 324
of LocalTime 327
of ZonedDateTime 331
isAfterLast method 296
of ResultSet 299
isBefore method
of LocalDate 324
of LocalTime 327
of ZonedDateTime 331
isBeforeFirst method 296
of ResultSet 299
isCellEditable method
of AbstractCellEditor 607
of AbstractTableModel 604
of CellEditor 613
of DefaultTableModel 604
of TableModel 588, 604
isCharacters method
of XMLStreamReader 192
isClosed method
of ResultSet 278
of Socket 216
of Statement 278
isConnected method
of Socket 216
isDefaultButton method
of JButton 572
isDirectory method
of BasicFileAttributes 95
of Files 94
of ZipEntry 86
isEchoOn method
of PasswordCallback 418
isEditable method
of JComboBox 524
of JTextComponent 507
isEmpty method
of Optional 18
isEnabled method 477
of Action 481
isEndElement method
of XMLStreamReader 192
isExecutable method 93
of Files 94
isFirst method 296
of ResultSet 299
isGroupingUsed method
of NumberFormat 349
isHidden method 93
of Files 94
isIgnoringElementContentWhitespace method
of DocumentBuilderFactory 170
isInputShutdown method
of Socket 224
isLast method 296
of ResultSet 299
isLeaf method
of DefaultTreeModel 621
of TreeModel 622, 641, 646
of TreeNode 621, 622
isLeapYear method
of LocalDate 324
isNamespaceAware method
of DocumentBuilderFactory 185
of SAXParserFactory 188
isNegative method
of Duration 321
isNullable method
of ParameterMetaData 309
ISO 216 372
ISO 3166-1 341
ISO 4217 351, 352
ISO 639-1 341
ISO 8601 292
ISO 8859-1 67, 75
isOther method
of BasicFileAttributes 95
isOutputShutdown method
of Socket 224
isParseIntegerOnly method

of NumberFormat 349
isPopupTrigger method
of JPopupMenu 536
of MouseEvent 536
isPresent method 17, 20
of Optional 18
isReadable method 93
of Files 94
isRegularFile method
of BasicFileAttributes 95
of Files 93, 94
isRejecting method
of Gatherer.Downstream 46
isResizable method
of Frame 449
isSelected method
of AbstractButton 535
of JCheckBox 514, 516
isShared method 110
isStartElement method
of XMLStreamReader 192
isSymbolicLink method
of BasicFileAttributes 95
of Files 93, 94
isTemporary method 489
isTraversable method 575
of FileView 579
isUnresolved method
of InetSocketAddress 227
isValidating method
of DocumentBuilderFactory 170
of SAXParserFactory 188
isVisible method
of Component 449
isWhiteSpace method
of XMLStreamReader 192
isWritable method 93
of Files 94
isZero method
of Duration 321
item method
of NamedNodeMap 162
of NodeList 156, 162, 168
ItemEvent class 488
getXxx methods 488
ItemListener interface, itemStateChanged method 488
ItemSelectable interface, getSelectedObjects method 517
Iterable interface 8, 98, 380
spliterator method 8
iterate method
of Stream 4, 7, 12, 47
Iterator interface 276
iterator method
of BaseStream 22
of SQLException 279, 280
of Stream 21
Iterators 21
converting to streams 5, 52

J

J (long), type code 119
JAAS 409
configuration files in 410
login modules in 412
JAR files
adding to class path 399
automatic registration in 275
for plugins 399
manifest of 85
resources in 369
signing 426, 430
jar: scheme 231
Jarsigner 426
JarXxxStream classes 85
Java
-noverify option 408
internationalization support in 339
platform-independent 78
specifying locales in 343
versions of 443, 546
vs. SQL 286
Java 2D API 647
affine transformations in 676
colors in 698
constructive area geometry operations in 663
features supported in 647
filters in 703
paint in 671
printing in 720
rendering pipeline in 647
sample values in 697
shape classes in 650, 654
strokes in 664
transparency in 680, 682
Java 2D library 454
floating-point coordinates in 454
Java Bug Database 704
Java Development Kit (JDK)
documentation in 481
fonts shipped with 462
Java EE 265
Java look-and-feel 479
Java Native Interface 741
Java Plug-in, loading signed code 430
Java Virtual Machine Specification 407
java.awt.geom package 125
java.base package 398
java.datatransfer package 398
java.desktop package 398
java.instrument package 398
java.io package 75
java.logging package 398
java.management package 398
java.management.rmi package 398
java.naming package 398
java.net package
socket connections in 215
supporting IPv6 addresses in 216
URLs vs. URIs in 231
java.net.http package 247
java.nio package 224
memory mapping in 102
java.prefs package 398
java.rmi package 398
java.security package 419
java.security.sasl package 398
java.sql package 336

java.time package 353, 354
java.xml package 398
JavaBeans 577
Javac
  byte order mark in 368
  passing arguments to 380
JavaCompiler interface 386
  getStandardFileManager method 386
  getTask method 380, 386
JavaCompiler.CompilationTask interface 386
  call method 386
JavaFileObject interface 380
JavaFX 444
JavaScript 387, 394
javax.imageio package 688
javax.sql package 315
javax.sql.rowset package 300
javax.swing package 446
javax.xml.catalog.files system property 164
JAXP (Java API for XML Processing) 154
JAXP (Java API for XML Processing) library 400
JButton class 470, 474, 479, 500, 572
  Constructor 474
  isDefaultButton method 572
Jchar 744
JCheckBox class 514, 516, 605
  Constructor 516
  is/setSelected methods 514
  isSelected method 516
  setSelected method 516
JCheckBoxMenuItem class 534
  Constructor 534
JComboBox class 488, 522, 524, 605
  addItem method 523, 524
  getItemAt method 522
  getSelectedItem method 522, 524
  insertItemAt method 523, 524
  isEditable method 524
  removeAllItems method 524
  removeAllItems, removeItemAt methods 523
  removeItem method 523, 524
  removeItemAt method 524
  setEditable method 522, 524
  setModel method 523
JComponent class 450, 453, 466, 482, 509, 521, 536, 546, 572, 623
  action maps 480
  get/setComponentPopupMenu methods 536
  get/setInheritsPopupMenu methods 536
  getActionMap method 482
  getComponentPopupMenu method 536
  getFontMetrics method 464, 466
  getInheritsPopupMenu method 536
  getInputMap method 480, 482
  getRootPane method 568, 572
  input maps 480, 481
  paint method 603, 647
  paintComponent method 450, 452, 453, 464, 467, 647
  putClientProperty method 618, 623
  revalidate method 508, 509
  setBorder method 520, 521
  setComponentPopupMenu method 536
  setFont method 509
  setInheritsPopupMenu method 536
  setToolTipText method 546
JDBC 263, 316
  configuration of 271
  debugging 274
  design of 263
  specification for 264
  tracing 274
  uses of 265
  versions of 263
JDBC API Tutorial and Reference (Fisher et al.) 298, 314
JDBC drivers
  escape syntax in 291
  JAR files for 272
  registering classes for 274
  scrollable/updatable result sets in 295
  types of 264
JDBC/ODBC bridge 264
JdbcRowSet interface 300
JDialog class 563, 566
  JDialog method 566
  setDefaultCloseOperation method 565
  setVisible method 564, 566, 567
JDialog method
  of JDialog 566
JDK
  DOM parser 154
  keytool program 424
  serialver program 129
  SunJCE ciphers 432
jdk.incubator.http package 247
JEditorPane class 512
JEP 290, JEP 415 133
jextract tool 756
JFileChooser class 572, 577
  addChoosableFileFilter method 578
  Constructor 577
  getSelectedFile method 578
  getSelectedFile/Files methods 574
  getSelectedFiles method 578
  resetChoosableFileFilters method 578
  resetChoosableFilters method 575
  setAcceptAllFileFilterUsed method 575, 578
  setAccessory method 578
  setCurrentDirectory method 572, 577
  setFileFilter method 575, 578
  setFileSelectionMode method 573, 578
  setFileView method 575, 576, 578
  setMultiSelectionEnabled method 573, 577
  setSelectedFile method 577
  setSelectedFile/Files methods 573
  setSelectedFiles method 577
  showDialog method 568, 572, 574, 578
  showOpenDialog method 578
  showSaveDialog method 578
  show*Xxx*Dialog methods 572, 573
JFrame class 445, 448, 453, 502, 533
  add method 453
  internal structure of 450, 451
  setJMenuBar method 530, 533
JLabel class 509, 510, 576, 631, 632
  Constructor 510
  getIcon method 510
  getText method 510
  setIcon method 510

setText method 510
JMenu class 532
add method 531, 532
addSeparator method 531, 532
Constructor 532
insert method 532
insertSeparator method 532
remove method 532
JMenuBar class 530
JMenuItem class 532, 533, 538, 540
Constructor 532, 533, 538
setAccelerator method 537, 538
setEnabled method 539, 540
setIcon method 533
JMOD files 399
JNDI service 315, 400
JndiLoginModule 410
JNI 741
JNICALL, JNIEXPORT macros 743
JobAttributes class (obsolete) 738
Join styles 665
Joining 23
joining method
of Collectors 24
JoinRowSet interface 300
JOptionPane class 559, 562
message types 559
showConfirmDialog method 559, 562
showInputDialog method 559, 560, 563
showInternalConfirmDialog method 562
showInternalInputDialog method 563
showInternalMessageDialog method 562
showInternalOptionDialog method 563
showMessageDialog method 559, 562
showOptionDialog method 559, 561, 563
JPanel class 501, 505, 715
JPasswordField class 510
Constructor 510
getPassword method 510
setEchoChar method 510
JPEG 688
image manipulations on 706
printing 726
reading 689
JPopupMenu class 535, 536
isPopupTrigger method 536
show method 536
JRadioButton class 516, 519
Constructor 519
JRadioButtonMenuItem class 535
Constructor 535
JRootPane class 572
setDefaultButton method 572
JRootPane class, setDefaultButton method 568
JScrollbar 488
JScrollPane class 513, 582
Constructor 513
JSlider class 525, 530
Constructor 530
setInverted method 527
setLabelTable method 526, 530
setMajorTickSpacing method 530
setMinorTickSpacing method 530
setPaintLabels method 530
setPaintLabels, setPaintTicks, setSnapToTicks methods 526
setPaintTicks method 530
setPaintTrack method 527, 530
setSnapToTicks method 530
JSP 383
Jstring 745
JTable class 581, 585, 599, 612
addColumn method 596, 600
Constructor 585
convertColumnIndexToModel method 600
convertRowIndexToModel method 600
convert*Xxx*IndexToModel methods 592
default rendering actions 589
getAutoCreateRowSorter method 585
getCellRenderer method 604
getCellSelectionEnabled method 600
getColumnModel method 590, 599
getColumnSelectionAllowed method 600
getDefaultEditor method 612
getDefaultRenderer method 604, 612
getFillsViewportHeight method 585
getRowHeight method 600
getRowMargin method 600
getRowSelectionAllowed method 600
getSelectedColumns method 592, 596
getSelectedRows method 592
getSelectionModel method 600
getTableHeader method 584
installing cell editors automatically 605
moveColumn method 596, 600
print method 584, 585
removeColumn method 596, 600
resize modes of 591
setAutoCreateRowSorter method 583, 585, 592
setAutoResizeMode method 591, 600
setCellSelectionEnabled method 592, 600
setColumnSelectionAllowed method 592, 600
setDefaultRenderer method 604
setFillsViewportHeight method 585
setRowHeight method 600
setRowHeight, setRowMargin methods 591
setRowMargin method 600
setRowSelectionAllowed method 592, 600
setRowSorter method 592, 600
JTextArea class 511, 513
append method 513
Constructor 513
setColumns method 513
setColumns, setRows methods 511
setLineWrap method 511, 513
setRows method 513
setTabSize method 513
setWrapStyleWord method 513
JTextComponent
getText method 508
setText method 508
JTextComponent class 507
getText method 507
isEditable method 507
setEditable method 507
setText method 507
JTextField class 488, 507, 508, 605
Constructor 508

getColumns method 508
setColumns method 507, 508
JToolBar class 544, 545, 546
add method 546
add, addSeparator methods 544
addSeparator method 546
Constructor 546
JTree class 613, 622, 628, 639
addTreeSelectionListener method 634
constructor 614, 622
getLastSelectedPathComponent method 624, 628
getSelectionPath method 623, 628, 639
getSelectionPaths method 635, 639
identifying nodes 623
makeVisible method 625, 628
scrollPathToVisible method 625, 629
setEditable method 626
setRootVisible method 620, 622
setShowsRootHandles method 620, 622
JVM
class files in 398
class loaders in 398
specification for 407

K

Kerberos protocol 409
Kernel class 706, 710
Constructor 710
Kernel, of a convolution 705
Keyboard
associating with actions 479
focus of 479
mnemonics for 536
Keyboard, reading from 58
KeyEvent class 488
getKey*Xxx*, isActionKey methods 489
KeyGenerator class 433, 436
generateKey method 433, 437
getInstance method 436
init method 437
KeyListener interface, key*Xxx* methods 489
KeyPairGenerator class 438
keys method
of Preferences 491, 494
KeyStoreLoginModule 410
Keystores 424
KeyStroke class 481
getKeyStroke method 481
KeyStroke class, getKeyStroke method 479
Keytool 424
Krb5LoginModule 410

L

L (object), type code 119
Labels
for components 509
for slider ticks 526
Lambda expressions 9
for event listeners 474
Landscape orientation 676
Language codes 30, 341
last method
of ResultSet 296, 299
lastAccessTime method
of BasicFileAttributes 95
lastDayOfMonth method
of TemporalAdjusters 326
lastDayOfYear method
of TemporalAdjusters 326
lastInMonth method
of TemporalAdjusters 326
lastModifiedTime method
of BasicFileAttributes 95
Layout algorithm 719
Layout management 501
border 504
box 546
custom 555
flow 501
grid 506
grid bag 546
group 546
sophisticated 546
spring 546
layoutContainer method
of LayoutManager 558
LayoutManager interface 558
addLayoutComponent method 558
designing custom 555
layoutContainer method 558
minimumLayoutSize method 558
preferredLayoutSize method 558
removeLayoutComponent method 558
LayoutManager2 interface 556
layoutPages method 719
Lazy operations 2, 6, 12, 144
LCD displays 698
LD_LIBRARY_PATH 746
Leading 463
Leap seconds 317
Leap years 322
Learn SQL The Hard Way (Shaw) 266
Learning SQL (Beaulieu) 266
Leaves 614, 621, 641
icons for 621, 631
Legacy APIs 88, 336
Legacy data, converting into XML 205
length method
of Blob 291
of CharSequence 64
of Clob 291
of RandomAccessFile 81, 84
lib/ext directory 399
libraryLookup method
of SymbolLookup 759
LIKE 269
LIKE (SQL) 292
limit method
of Buffer 106
of Stream 11, 12, 52
Line feed 70, 367
in e-mails 259
in regular expressions 139
Line2D class 454, 457, 650
Line2D.Double class 460
Constructor 460

Line2D.Double, Line2D.Float classes 651
LineBorder class 520, 521
  LineBorder method 521
LineBorder method
  of LineBorder 521
LineMetrics class 464, 466
  getAscent method 466
  getDescent method 466
  getHeight method 466
  getLeading method 466
Lines 454
  constructing 457
lines method
  of Files 5, 8, 53, 89
  of String 8
lineTo method 655
  of Path2D.Float 663
Linker interface 747, 753, 758, 761
  canonicalLayouts method 758
  defaultLookup method 758
  downcallHandle method 759
  nativeLinker method 758
  upcallStub method 761
Linker.Option interface 765
  captureCallState method 765
  captureStateLayout method 765
  firstVariadicArg method 765
Linux
  library path in 746
  pop-up menus in 536
  using GNU C compiler 745
list method
  of Files 95
Listener interfaces 469
Listener objects 469
ListResourceBundle class 372
Lists, converting to streams 52
ListSelectionModel interface 601
  setSelectionMode method 592, 601
Little-endian order 74, 78, 103, 368
loadClass method
  of ClassLoader 400, 401
  of URLClassLoader 399
Loader lookup 754
loadLibrary method 745
  of System 747
LOBs 290
  placing in database 290
  reading 290
Local hosts 217
Local names 184
LocalDate class 323, 326, 336, 356
  datesUntil method 322, 324
  format method 353
  getDayOfMonth method 324
  getDayOfWeek method 324
  getDayOf*Xxx* methods 322
  getDayOfYear method 324
  getMonth method 324
  getMonthValue method 324
  getYear method 324
  isAfter method 324
  isBefore method 324
  isLeapYear method 324
  legacy classes and 336
  minus method 324
  minus, minus*Xxx* methods 322
  minus*Xxx* method 324
  now method 321, 323
  of method 321, 324, 325
  parse method 336, 354, 356
  plus method 324
  plus, plus*Xxx* methods 321, 322
  plus*Xxx* method 324
  toLocalDate method 337
  until method 322, 324
  weekends in 322
  with method 325, 326
  withDayOfMonth method 324
  withDayOfYear method 324
  withMonth method 324
  withYear method 324
LocalDateTime class 327, 356
  atZone method 328
  format method 353
  legacy classes and 336
  parse method 354, 356
  toLocalDateTime method 337
Locale class 340, 345
  availableLocales method 27
  forLanguageTag method 342, 345
  getCountry method 30, 345
  getDefault method 343, 345
  getDisplayCountry method 345
  getDisplayCountry, getDisplayLanguage methods 27
  getDisplayLanguage method 345
  getDisplayName method 344, 345, 347
  getInstance method 357
  getISOCountries method 343, 345
  getISOLanguages method 343
  getLanguage method 345
  Locale.of method 345
  predefined objects 342
  setDefault method 343, 345
  toLanguageTag method 342, 345
  toString method 345
Locale.of method
  of Locale 345
Locales 339
  current 363
  date and time formatting in 353
  default 332, 343
  dictionary ordering in 357
  display names of 27, 344
  formatting styles for 333
  grouping 30
  mapping names of 27
  numbers in 340, 346
  predefined 342
  resources bundles and 369
  variants in 341, 370
LocalTime class 326, 327, 356
  method 327
  format method 353
  getHour method 327
  getMinute method 327
  getNano method 327
  getSecond method 327

isAfter method 327
isBefore method 327
legacy classes and 336
minus method 327
minus, minus*Xxx* methods 326
now method 326, 327
of method 326, 327
parse method 354, 356
plus method 327
plus, plus*Xxx* methods 326
toLocalTime method 337
toNanoOfDay method 327
toSecondOfDay method 327
with*Xxx* method 327
lock method
of FileChannel 110, 112, 113
Locks 110
for the tail portion of a file 110
shared 110
unlocking 110
logging.properties file 250
login method
of LoginContext 411
of LoginModule 418
LoginContext class 409, 411
Constructor 411
getSubject method 411
login method 411
login, logout methods 409
logout method 411
LoginModule interface 418
abort method 418
commit method 418
documentation for 414
initialize method 418
login method 418
logout method 418
Logins
committed 414
modules for 410
custom 412
separating from action code 413
logout method
of LoginContext 411
of LoginModule 418
Long class
MAX_VALUE constant 110
LONG NVARCHAR 313
long type
printing 70
streams of 46
type code for 119
writing in binary format 78
LONG VARCHAR 313
LongBuffer class 107
longs method
of Random 47
of RandomGenerator 50
LongStream interface 46, 49
average method 48, 49
boxed method 47, 49
mapToLong method 47
max method 49
max, min methods 48
min method 49
of method 49
range method 49
range, rangeClosed methods 47
rangeClosed method 49
sum method 49
sum, summaryStatistics methods 48
summaryStatistics method 49
toArray method 47, 49
LongSummaryStatistics class 23, 25, 48, 51
getAverage method 25, 51
getCount method 25, 51
getMax method 25, 51
getMin method 25, 51
getSum method 25, 51
Look-and-feel
appearance of buttons in 498
displaying trees in 618
handles for subtrees in 631
pluggable 575
selecting multiple nodes in 635
lookingAt method
of Matcher 147
lookup method
of MethodHandles 762
LookupOp class 703, 704, 709
Constructor 709
lookupPrintServices method 727
of PrintServiceLookup 728
lookupStreamPrintServiceFactories method
of StreamPrintServiceFactory 729
LookupTable class 704
Low-level events 488
LSB (least significant byte) 78
LSOutput 194
LSSerializer 194
writeToString method 194

M

Mac OS X
character encodings in 366
resources in 369
Mac Roman 366
main method 398
makeShape method 656
makeVisible method 625
of JTree 628
Mandelbrot set 699
Mangling names 744
Manifest files 85
map method
of FileChannel 102, 106
of HttpHeaders 249, 254
of Optional 16
of Stream 9, 10
mapConcurrent method
of Gatherers 45
mapMulti method
of Stream 11
mapping method
of Collectors 32, 35
Maps
concurrent 28

of stream elements 27, 52
mapToInt method 39
mapTo*Xxx* methods
of *Xxx*Stream 47
mark method
of Buffer 108, 109
of InputStream 59
markSupported method
of InputStream 59
Mastering Regular Expressions (Friedl) 139
match attribute (XSLT) 203
Matcher class 147
find method 147
hasMatch method 147
lookingAt method 147
matches method 147
of Pattern 140
quoteReplacement method 147
replaceAll method 145, 147
replaceFirst method 146, 147
results method 147
matcher method
of Pattern 140, 146
matches method
of Matcher 147
of PathMatcher 98
of Pattern 140
MatchResult interface 141, 147
end method 148
group method 147
namedGroups method 148
start method 148
Matisse 546
Matrices, transformations of 675, 676
Max
of primitive streams 48
max method
of DoubleStream 50
of IntStream 49
of LongStream 49
of Stream 13, 14
MAX_VALUE 110
maxBy method 31
of Collectors 35
maxoccurs attribute (XSLT) 172
MD5 420
Memory addresses, vs. serial numbers 115
Memory layout 751
hierarchy 752
Memory layouts
canonical 751
Memory mapping of files 101
Memory segment 749
allocating 749
off-heap 748, 749
on-heap 749
reading 750
writing 750
MemoryLayout interface 753, 765
byteOffset method 753
byteOffsetHandle method 765
sequenceLayout method 753
structLayout method 753
unionLayout method 753
varHandle method 765
MemoryLayout.PathElement interface 753, 765
dereferenceElement method 765
groupElement method 753
sequenceElement method 753, 765
MemorySegment interface 750, 759
get method 751
getAtIndex method 751
getString method 750
reinterpret method 759
set method 751
setAtIndex method 751
toArray method 751
menuCanceled method
of MenuListener 540
menuDeselected method
of MenuListener 540
MenuListener interface 539, 540
menuCanceled method 540
menuDeselected method 540
menuSelected method 540
menu*Xxx* methods 539
Menus 530
accelerators for 537, 538
checkboxes in 534
icons in 533
keyboard mnemonics for 536
menu bar in 530, 531
menu items in 530
enabling/disabling 539
pop-up 535
radio buttons in 534
submenus in 530, 531
menuSelected method
of MenuListener 540
Message digests 419
MessageDigest class 421
digest method 420, 421
extending 420
getInstance method 420, 421
reset method 421
update method 420, 421
MessageFormat class 361, 363, 373
applyPattern method 363
Constructor 363
format method 362, 363
getLocale method 363
ignoring the first limit 365
setLocale method 363
Metadata 303
Metal look-and-feel
selecting multiple nodes in 635
trees in 617, 618
Method handle 755
downcall 755
Method verification errors 408
MethodHandle class 747, 759
invoke method 759
MethodHandle.Lookup 762
findStatic method 762
MethodHandles class 762
lookup method 762
Methods
do-nothing 475

mangling names of 744
native 741
protected 399
MethodType class 762
methodType method 762
methodType method
of MethodType 762
Microsoft
compiler 745
ODBC API 263
SQL Server 271
Visual Basic 469
Microsoft Windows. See Windows operating system
MIME 689
for print services 726
getting, of a file 89
MimeMessage class
methods of 260
Min
of primitive streams 48
min method
of DoubleStream 50
of IntStream 49
of LongStream 49
of Stream 13, 14
minBy method 31
of Collectors 35
minimumLayoutSize method
of LayoutManager 558
minoccurs attribute (XSLT) 172
minus method
of Duration 320
of Instant 320
of LocalDate 324
of LocalTime 327
of Period 325
of ZonedDateTime 330
minus, minus*Xxx* methods
of LocalDate 322
of LocalTime 326
minus*Xxx* method
of Duration 321
of Instant 320
of LocalDate 324
of Period 325
of ZonedDateTime 330
mismatch method
of Files 89, 90
MissingResourceException class 370
Miter join, miter limit 665
Mixed content 152, 166
Modality 559, 563
Model-view-controller 497
classes in 498
multiple views in 499
Modernist painting example 197
Modified UTF-8 79, 368
native code and 744
Modules
platform classes in 398
Month enumeration 321
getDisplayName method 332, 354
MonthDay class 323
Mouse events 482
MouseAdapter class 484
MouseEvent class 486, 488, 536
getClickCount method 482, 486, 489
getPoint method 486, 489
getX method 486
getX/Y methods 482, 489
getY method 486
isPopupTrigger method 536
translatePoint method 489
MouseHandler 484
MouseListener interface 483
mouseClicked method 482, 484, 489
mouseDragged method 483
mouseEntered/Exited methods 484, 489
mousePressed/Released methods 482, 489
MouseMotionHandler 484
MouseMotionListener interface 483, 484
mouseDragged method 489
mouseMoved method 483, 484, 489
MouseWheelEvent class 488
getScrollAmount, getWheelRotation methods 489
MouseWheelListener interface, mouseWheelMoved method 489
move method
of Files 91, 93
moveColumn method 596
of JTable 600
moveTo method 655
of Path2D.Float 663
moveToCurrentRow method
of ResultSet 299
moveToInsertRow method
of ResultSet 299
moveTo*Xxx*Row methods (ResultSet) 297
MSB (most significant byte) 78
Multiple-page printing 720
multipliedBy method
of Duration 321
MutableTreeNode interface 622
implementing 615
setUserObject method 616, 622
MySQL 271

N

NameCallback class 413, 417
Constructor 417
getDefaultName method 417
getName method 417
getPrompt method 417
setName method 417
namedGroups method
of MatchResult 148
of Pattern 147
NamedNodeMap interface 162
getLength method 162
item method 162
Namespaces 182
activating processing of 173
aliases (prefixes) for 170, 183
of attributes 183
of child elements 182
using class loaders as 401
National character strings 314
National Institute of Standards and Technology 212, 420

native keyword 742
Native methods 741
  calling from Java programs 742
  compiling 745
  garbage collection and 745
  linking to Java 746
  naming 742, 743
  overloading 743
  reasons to use 741
  static 742
  strings in 744
native.encoding package 76
nativeLinker method
  of Linker 758
NCHAR, NCLOB data types (SQL) 313, 314
negated method
  of Duration 321
Nervous text applet 418
net.properties package 249
NetBeans
  Matisse 546
Netscape
  IFC library 444
Networking 211, 261
  connecting to a server 211
  debugging 211
  getting web data 231
  HTTP client for 247
  implementing servers 218
  sending e-mails 259
newBufferedReader method
  of Files 90
newBufferedWriter method
  of Files 90
newBuffered*Xxx* methods
  of Files 90
newBuilder method
  of HttpClient 247, 253
  of HttpRequest 249, 254
newDirectoryStream method 98
  of Files 100
newDocument method 192, 206
  of DocumentBuilder 194
newDocumentBuilder method 154, 193
  of DocumentBuilderFactory 160
newFactory method 300
  of RowSetProvider 303
newFileSystem method 100
  of FileSystems 101
newHttpClient method 247
  of HttpClient 253
newInputStream method
  of Channels 227
  of Files 90
newInstance method
  of DocumentBuilderFactory 154, 160, 183
  of SAXParserFactory 186, 188
  of TransformerFactory 194, 195
  of XMLInputFactory 191
  of XMLOutputFactory 196, 200
  of XPathFactory 179, 181
newNSInstance method 184, 187
  of DocumentBuilderFactory 185
  of SAXParserFactory 188
newOutputStream method
  of Channels 225, 227
  of Files 90
newSAXParser method 186
  of SAXParserFactory 188
NewStringUTF 745
newTransformer method 194
  of TransformerFactory 195, 209
newXpath method
  of XPathFactory 181
next method
  of ResultSet 276, 278, 294
  of TemporalAdjusters 326
  of XMLStreamReader 192
nextElement method 629
nextOrSame method 325
  of TemporalAdjusters 326
nextPage method 301
  of CachedRowSet 303
NMTOKEN, NMTOKENS attribute types (DTDs) 166, 167
No-argument constructors 116, 132
Node interface 161, 184, 195
  appendChild method 193, 195
  getAttributes method 158, 161
  getChildNodes method 156, 161
  getFirstChild method 157, 161
  getLastChild method 158, 161
  getLocalName method 184
  getNamespaceURI method 184
  getNextSibling method 158, 161
  getNodeName method 161
  getNodeValue method 161
  getNode*Xxx* methods 158, 184
  getParentNode method 161
  getPreviousSibling method 161
  subinterfaces of 155
node method
  of Preferences 491, 494
nodeChanged method 625
  of DefaultTreeModel 629
NodeList interface 156, 162
  getLength method 156, 162
  item method 156, 162, 168
Nodes class 614
  adding 624
  child 614, 616
  collapsed 626
  connecting lines for 618, 619
  currently selected 623
  editing 626, 641
  enumerating 629
  expanding 625, 626
  handles for 617, 620, 631
  highlighting 632
  identifying, by tree paths 623
  making visible 625
  parent 614, 616
  removing 624
  rendering 621, 631
  root 614, 620
  row positions of 624
  searching 630, 635
  selecting 634
  user objects for 615, 625, 630, 635

nodesChanged method
of DefaultTreeModel 629
Nondeterministic parsing 166
noneMatch method 14
of Stream 14
Noninterference, of stream operations 6
Normalization 358
normalize method
of Normalizer 361
of Path 88, 89
Normalized color values 697
Normalizer class 359, 361
normalize method 361
NoSuchElementException class 17
notFilter method 595
of RowFilter 602
NotSerializableException class 125
now method
of Instant 318, 320
of LocalDate 321, 323
of LocalTime 326, 327
of ZonedDateTime 330
NTLoginModule 410
nullInputStream method
of InputStream 59
nullOutputStream method
of OutputStream 60
NullPointerException class
vs. Optional 13
Number class
doubleValue method 346
intValue method 346
numberFilter method 594
of RowFilter 602
NumberFormat class 346, 348
format method 347, 349
getAvailableLocales method 343, 347, 348
getCompactNumberInstance method 349
getCurrencyInstance method 346, 348, 352
getMaximumFractionDigits method 349
getMaximumIntegerDigits method 349
getMinimumFractionDigits method 349
getMinimumIntegerDigits method 349
getNumberInstance method 346, 348
getPercentInstance method 346, 348
isGroupingUsed method 349
isParseIntegerOnly method 349
parse method 346, 347, 349
setCurrency method 352
setGroupingUsed method 349
setMaximumFractionDigits method 349
setMaximumIntegerDigits method 349
setMinimumFractionDigits method 349
setMinimumIntegerDigits method 349
setParseIntegerOnly method 349
Numbers
filtering 594
floating-point 340, 346
formatting 340, 346, 361
supported locales for 347
from grouped elements 31
in regular expressions 137
printing 70
random 4, 11, 47, 433
reading
from files 64
from ZIP archives 66
using locales 346
storing in memory 78
writing in binary format 78
NUMERIC 271, 313
NVARCHAR 313

O

Object class
clone method 114, 133
Object inspection tree 639
Object serialization 113
cloning with 133
file format for 118
serial numbers for 115
ObjectInputStream class 114, 117
Constructor 117
readObject method 114, 118, 126
ObjectInputValidation interface 132
ObjectOutputStream class 113, 114, 117
Constructor 117
defaultWriteObject method 126
writeObject method 113, 117, 126
Objects
cloning 133
converting to streams 4, 20
deserialized 128
fingerprints of 119
printing 70
reading from an input stream 114
saving
in output streams 113, 115
in text format 71
serializable 113
transmitting over network 115
type code for 119
versioning 129
ODBC 263, 265
of method
of Collector 38
of DoubleStream 49
of FunctionDescriptor 759
of Gatherer 46
of Gatherer.Integrator 46
of IntStream 47, 49
of LocalDate 321, 324, 325
of LocalTime 326, 327
of LongStream 49
of Optional 18
of OptionalDouble 51
of OptionalInt 51
of OptionalLong 51
of Path 87, 88
of Period 325
of Stream 3, 7
of ZonedDateTime 328, 330
of ZoneId 328
ofAuto method
of Arena 749
ofConfined method
of Arena 749

ofDateAdjuster method 325
ofDays method
  of Period 322, 329
Off-heap memory segment 748
ofFile, ofFileDownload methods
  of BodyHandlers 249
OffsetDateTime class 329
ofInstant method
  of ZonedDateTime 330
ofLocalizedDate method
  of DateTimeFormatter 335, 356
ofLocalizedDateTime method
  of DateTimeFormatter 335, 356
ofLocalizedPattern method
  of DateTimeFormatter 336
ofLocalizedTime method
  of DateTimeFormatter 335, 356
ofLocalized*Xxx* methods
  of DateTimeFormatter 332, 353
ofMonths method 322
ofNullable method
  of Optional 18
  of Stream 4, 7, 20
ofPattern method 333
  of DateTimeFormatter 336
ofSequential method
  of Gatherer 46
ofShared method
  of Arena 749
ofString method
  of BodyHandlers, BodyPublishers 248, 249
ofVoid method
  of FunctionDescriptor 759
of*Xxx* method
  of Duration 320
  of Period 325
oj (SQL escapes) 292
open keyword
  of FileChannel 102
  of SocketChannel 224
open method
  of FileChannel 106
  of SocketChannel 227
Open path element 763
openConnection method 233
  of URL 238
openOutputStream method
  of SimpleJavaFileObject 387
openStream method 64, 231
  of URL 238
Operating system
  character encodings in 76, 366
  default locales in 343
  paths in 64, 87
  resources in 369
Operations
  associative 39
  lazy 2, 6, 12, 144
  stateless 51
Option panes 559
Optional class 13, 15, 16, 18, 19, 21
  creating values of 18
  empty method 18
  filter method 16
  flatMap method 18, 19, 21
  for empty streams 38, 39
  get method 17, 18, 20
  ifPresent method 15, 16
  ifPresentOrElse method 15, 16
  isEmpty method 18
  isPresent method 17, 18, 20
  map method 16
  of method 18
  of, ofNullable methods 18
  ofNullable method 18
  or method 16, 17
  orElse method 13, 15
  orElseGet method 15
  orElseThrow method 15, 18
  stream method 19
optional option (JAAS configuration) 410
OptionalDouble class 51
  getAs*Xxx* method 51
  ifPresent method 51
  of method 51
  orElse method 51
  orElseGet method 51
OptionalInt class 51
  getAs*Xxx* method 51
  ifPresent method 51
  of method 51
  orElse method 51
  orElseGet method 51
OptionalInt, OptionalDouble, OptionalLong classes 48
OptionalLong class 51
  getAs*Xxx* method 51
  ifPresent method 51
  of method 51
  orElse method 51
  orElseGet method 51
or method
  of Optional 17
Oracle
  JDBC 263, 271
  JVM implementation 76
ORDER BY 277
order method
  of ByteBuffer 103, 107
orElse method
  of Optional 15
  of OptionalDouble 51
  of OptionalInt 51
  of OptionalLong 51
orElseGet method
  of Optional 15
  of OptionalDouble 51
  of OptionalInt 51
  of OptionalLong 51
orElseThrow method
  of Optional 15, 18
orFilter method 595
  of RowFilter 602
org.w3c.dom package 154
Outer joins 292
output element (XSLT) 203
Output streams 57
  buffered 65
  byte processing in 65

byte-oriented 57
closing 58, 223
filters for 64
hierarchy of 60
objects in 113
redirecting 391
Unicode and 57
OutputStream class 57, 60, 62, 196
close method 60
flush method 58, 60
nullOutputStream method 60
write method 58, 60
OutputStreamWriter class 67
OverlappingFileLockException class 111
Overloading 743
Owner frame 563
Owner thread
of arena 748

P

pack method
of Window 452, 454
Packages
avoiding name clashes with 401
Packets 215
Padding schemes 432
PaddingLayout interface 752
Page setup dialog box 714, 715
Pageable interface
implementing 718
objects, printing 726
PageAttributes class (obsolete) 738
pageDialog method 714, 715
of PrinterJob 718
PageFormat class 718
getHeight method 713, 718
getImageableHeight method 718
getImageableWidth method 718
getImageableX method 718
getImageable*Xxx* methods 713
getImageableY method 718
getOrientation method 718
getWidth method 713, 718
Pages
measurements of 713
multiple, printing 718
orientation of 676, 714
Paint interface 671
paint method
of JComponent 603, 647
paintComponent method 450, 452, 464, 467
of JComponent 453, 647
of StrokePanel 666
overriding 487
Paper class
margins of 713
sizes of 372, 713
parallel method
of BaseStream 55
parallelStream method 2, 51
of Collection 3, 55
ParameterMetaData interface 309
getParameterClassName method 309
getParameterCount method 309
getParameterMode method 309
getParameterType method 309
getParameterTypeName method 309
getPrecision method 310
getScale method 310
isNullable method 309
Parent classes. See Superclasses
Parent nodes 614, 616
parse method
of DateTimeFormatter 334
of DocumentBuilder 161
of LocalDate 336, 354, 356
of LocalDateTime 356
of LocalTime 356
of NumberFormat 346, 347, 349
of SAXParser 186, 188
of XMLReader 210
of ZonedDateTime 336, 354, 356
Parsed character data 165
ParseException class 347
Parsers 154
checking uniqueness of IDs in 167, 174
pull 189
validating in 162
Parsing 154
nondeterministic 166
with XML Schema 173
partitioningBy method 30, 33
of Collectors 31
Password-protected resources 235
PasswordCallback class 413, 418
Constructor 418
getPassword method 418
getPrompt method 418
isEchoOn method 418
setPassword method 418
PasswordChooser 567
Passwords
dialog box for 567
fields for 510
Path element
open 763
Path interface 86, 88
getFileName method 89
getParent method 89
getRoot method 89
get*Xxx* methods 88
normalize method 88, 89
of method 87, 88
relativize method 88
resolve method 88
resolve, resolveSibling methods 87
resolveSibling method 88
toAbsolutePath method 89
toAbsolutePath, toFile methods 88
toFile method 89
Path2D class 663
append method 655, 663, 679
closePath method 655, 663
Path2D.Double class 651
Path2D.Float class 651, 663
curveTo method 663
curveTo, lineTo, moveTo, quadTo methods 655

lineTo method 663
moveTo method 663
quadTo method 663
pathFromAncestorEnumeration method 630
PathMatcher interface 98
matches method 98
Paths (file system) 86
absolute vs. relative 64, 86, 88
checking properties of 93
filtering 96
relativizing 88
resolving 87
root component of 86
separators in 64, 86
Paths (graphics) 655, 656
Paths class 100
get method 87
Pattern class 8, 139, 146
asMatchPredicate, asPredicate methods 140
compile method 140, 146
flags 146
matcher method 140, 146
matches method 140
namedGroups method 147
split method 144, 147
splitAsStream method 4, 8, 144, 147
splitWithDelimiters method 147
Patterns 134
#PCDATA 165
PDF format 726
peek method
of Stream 12, 13
Percentages, formatting 346, 347
Performance
of encryption algorithms 438
of file operations 101
Period class 325
get*Xxx* method 325
minus method 325
minus*Xxx* method 325
of method 325
of*Xxx* method 325
of*Xxx* methods 322, 329
plus method 325
plus*Xxx* method 325
using for daylight savings time 329
with*Xxx* method 325
Perl programming language 139
Personal data 430
Pixels
affine transformations on 703
average value of 705
composing 680
interpolating 671, 703
sample values of 697
setting individual 697
Placeholders, in message formatting 365
Placeholders, in messages 362
Plugins, loading 399
plus method
of Duration 320
of Instant 320
of LocalDate 324
of LocalTime 327
of Period 325
of ZonedDateTime 330
plus, plus*Xxx* methods
of LocalDate 321, 322
of LocalTime 326
plus*Xxx* method
of Duration 321
of Instant 320
of LocalDate 324
of Period 325
of ZonedDateTime 330
PNG 688
printing 726
Point class 456
Point size 462
Point2D class 456, 651
Point2D.Double class 125, 456, 460, 651
Constructor 460
Point2D.Float class 456, 651
Points, in typography 713
Policy files 430
Polygons 650, 655
Pools, for parallel streams 53
Pop-up menus 535
populate method
of CachedRowSet 301, 303
Porter–Duff rules 681, 683
Portrait orientation 714
Ports 212
blocking 211
in URIs 232
position function (XPath) 204
POSIX-compliant file systems 94
PosixFileAttributes interface 94
POST 239, 241, 243, 244
building 248
POST method
of HttpRequest.Builder 248, 254
PostgreSQL 271
connecting to 274
drivers for 272
Postorder traversal 630
postOrderEnumeration method 630
of DefaultMutableTreeNode 633
PostScript
printing 726, 729
writing to 729
postVisitDirectory method
of FileVisitor 98
of SimpleFileVisitor 100
Predefined action table names 478
Predefined character classes 135, 137, 139
Predicate functions 30
Preferences
accessing 490
enumerating keys in 491
importing/exporting 492
Preferences class 490, 494
exportNode method 495
exportSubtree method 495
export*Xxx* methods 492
get method 494
get, get*DataType* methods 491
getBoolean method 494

getByteArray method 494
getDouble method 494
getFloat method 494
getInt method 494
getLong method 494
importPreferences method 492, 495
keys method 491, 494
node method 491, 494
platform-independency of 490
put method 495
putBoolean method 495
putByteArray method 495
putDouble method 495
putFloat method 495
putInt method 495
putLong method 495
system/userNodeForPackage methods 491
system/userRoot methods 490
systemNodeForPackage method 494
systemRoot method 494
userNodeForPackage method 494
userRoot method 494
preferredLayoutSize method
of LayoutManager 558
preOrderEnumeration method 630
of DefaultMutableTreeNode 633
Prepared statements 284
caching 286
executing 285
PreparedStatement interface 289, 309
clearParameters method 289
executeQuery method 289
executeUpdate method 289
executeXxx methods 285
getMetaData method 309
setXxx method 289
setXxx methods 285
prepareStatement method 285, 286, 295
of Connection 289, 298
Previous 295
previous method
of ResultSet 296, 299
of TemporalAdjusters 326
previousOrSame method
of TemporalAdjusters 326
previousPage method
of CachedRowSet 303
preVisitDirectory method
of FileVisitor 98
of SimpleFileVisitor 100
Primitive types
I/O in binary format in 61
streams of 39
Principal interface 412
getName method 412
Principals (logins) 410
Print dialog box 711
displaying page ranges in 712, 719
native 712, 715
print method
of DocPrintJob 728
of JTable 584, 585
of Printable 710, 717, 719
of PrinterJob 718
of PrintWriter 70, 71
Print services 726
document flavors for 726
for images 728
stream 729
print, println functions (JavaScript) 391
Printable interface 717
implementing 710, 715
objects, printing 726
print method 710, 717, 719
printDialog method
of PrinterJob 718
Printer graphics context 719
PrinterException class 712
PrinterJob class 717
defaultPage method 717
getPrinterJob method 711, 717
pageDialog method 714, 715, 718
print method 712, 718
printDialog method 711, 712, 718
setPageable method 719
setPrintable method 718
printf method
of PrintWriter 70, 71, 345
Printing
clipped areas 713
counting pages during 712
images 710
layout of 719
multipage documents 720
number of copies for 732
page orientation of 676, 714
paper sizes in 713
quality of 734
selecting settings for 711
starting 711
text 710
using
banding for 712
transformations for 720
Printing attributes 731
adding 734
categories of 733, 734
checking values of 734
hierarchy of 732
retrieving 734
PrintJob class (obsolete) 711
PrintJobAttribute interface 731
printing attributes of 734, 738
PrintJobAttributeSet interface 732
println method
of PrintWriter 70, 71, 72
of System.out 367
PrintQuality 734
PrintRequestAttribute interface 731
printing attributes of 734, 738
PrintRequestAttributeSet interface 711, 732
PrintService interface 728, 738
createPrintJob method 728
getAttributes method 738
getName method 728
PrintServiceAttribute interface 731
printing attributes of 734, 738
PrintServiceAttributeSet interface 732

PrintServiceLookup class 728
  lookupPrintServices method 727, 728
PrintStream class 70
PrintWriter class 69, 71
  checkError method 70, 71
  Constructor 71
  print method 70, 71
  printf method 70, 71, 345
  println method 70, 71, 72
Private keys 422, 438
probeContentType method 89
  of Files 90
Processing instructions 154
Programmer's Day 321
Programs. See Applications
Project Panama 747
Properties class 149, 447
Property files 149
  character encoding of 371
  for resources bundles 369
  for string resources 369
  for strings 370
  no passwords in 259
PropertyChangeListener interface 577
PUBLIC 194
Public key ciphers 422, 428, 438
  performance of 438
Public Key Cryptography Standard (PKCS) #5 432
Pull parsers 189
push method
  of Gatherer.Downstream 45
PushbackInputStream class 65, 67
  Constructor 67
  unread method 67
put method
  of Bindings 391
  of ByteBuffer 106, 107
  of CharBuffer 107
  of HttpRequest.Builder 254
  of Preferences 491, 495
  of ScriptEngine 390
  of ScriptEngineManager 390
putBoolean method
  of Preferences 495
putByteArray method
  of Preferences 495
putClientProperty method 618
  of JComponent 623
put*DataType* methods
  of Preferences 491
putDouble method
  of Preferences 495
putFloat method
  of Preferences 495
putInt method
  of Preferences 495
putLong method
  of Preferences 495
putNextEntry method 84
  of ZipOutputStream 85
putValue method 477
  of Action 481
put*Xxx* method
  of ByteBuffer 107
put*Xxx* methods
  of ByteBuffer 103

Q

QBE 268
QuadCurve2D class 650, 654
QuadCurve2D.Double class 651, 662
  Constructor 662
QuadCurve2D.Float class 651
Quadratic curves 654
quadTo method 655
  of Path2D.Float 663
Qualified names 184
Queries 269, 271
  by example 268
  executing 276, 284
  multiple 278
  populating row sets with results of 301
  preparing 284
  returning multiple results 293
quoteReplacement method
  of Matcher 147

R

R 387
Race conditions 52
Radio buttons 516, 534
Random class 433
  methods of 47
Random numbers, streams of 4, 11, 47
Random-access files 80
RandomAccessFile class 80, 84, 102, 106, 113
  Constructor 84
  getChannel method 106, 113
  getFilePointer method 80, 84
  length method 81, 84
  seek method 80, 84
RandomGenerator interface 50
  doubles method 50
  equiDoubles method 50
  ints method 50
  longs method 50
Randomness 433
range method
  of IntStream 49
  of LongStream 49
range, rangeClosed methods
  of *Xxx*Stream 47
rangeClosed method
  of IntStream 49
  of LongStream 49
Ranges, converting to streams 52
Raster class 701
  getDataElements method 698, 699, 701
  getPixel method 702
  getPixel, getPixels methods 698
  getPixels method 702
Raster images 688
  constructing from pixels 697
  filtering 703
  readers/writers for 688
read method

of CipherInputStream 438
of FileInputStream 58
of ImageIO 688, 694
of ImageReader 695
of InputStream 57, 59
of Readable 63
of ReadableByteChannel 109
of Reader 62
Readable interface 62, 63
read method 63
ReadableByteChannel 109
read method 109
ReadableByteChannel interface 224
readAllBytes method 58
of Files 90
of InputStream 59
readAllLines method
of Files 90
readAll*Xxx* methods
of Files 89
readAttributes method 94
of Files 94
readBoolean method 79
of DataInput 79
readByte method
of DataInput 79
readChar method 79, 81
of DataInput 79
readDouble method 79, 114, 126
of DataInput 79
Reader class 57, 62
read method 62
readExternal method 127, 128
readFixedString method 81, 82
readFloat method 79
of DataInput 79
readFully method
of DataInput 80
readInt method 79, 81, 114
of DataInput 79
readLong method 79
of DataInput 79
readNBytes method 58
of InputStream 59
readObject method
of HashSet 126
of ObjectInputStream 114, 118, 126
ReadOnlyBufferException class 102
readResolve method 128
readShort method 79
of DataInput 79
readString method 89
of Files 90
readThumbnail method
of ImageReader 695
readUTF method 79
of DataInput 80
REAL 271, 313
Records 71
deserializing 132
fixed-size 81, 82
ignoring serialVersionUID fields 131
Rectangle class 456
Rectangle2D class 454, 457, 650
Rectangle2D.Double class 455, 459
Constructor 459
Rectangle2D.Double, Rectangle2D.Float classes 651
Rectangle2D.Float class 455
Rectangles 454
drawing 454
filling with color 460
RectangularShape class 456, 459, 651
getCenterX method 459
getCenterX/Y methods 456
getCenterY method 459
getHeight method 459
getHeight/Width methods 456
getMaxX method 459
getMaxY method 459
getMinX method 459
getMinY method 459
getWidth method 459
getX method 459
getY method 459
Redirects, of URLs 244
reduce method
of Stream 38, 40
reducing method
of Collectors 33
Reductions 13, 38
ref attribute (XML Schema) 172
Reflection
accessing
protected methods 399
constructing class trees 635
enumerating fields from a variable 642
regexFilter method 594
of RowFilter 602
Regular expressions 134
escapes in 72
filtering 594
finding matches of 140
flags for 146
groups in 142
in DTDs 165, 166
splitting sequences with 4
reinterpret method
of MemorySegment 759
relative method
of ResultSet 296, 299
Relativization, in URLs 233
relativize method
of Path 88
releaseSavepoint method 311
of Connection 312
ReleaseStringUTFChars 745
reload method
of DefaultTreeModel 625, 629
remaining method
of Buffer 108, 109
remove method
of AttributeSet 738
of JMenu 532
removeAllItems method 523
of JComboBox 524
removeCellEditorListener method
of CellEditor 613
removeColumn method 596

of JTable 600
removeItem method 523
of JComboBox 524
removeItemAt method 523
of JComboBox 524
removeLayoutComponent method
of LayoutManager 558
removeNodeFromParent method 624
of DefaultTreeModel 629
removePropertyChangeListener method 477, 478
removeTreeModelListener method 641
of TreeModel 647
RenderableImage 726
Rendering
cells 602, 604
columns 588, 590
headers 604
nodes 631
Rendering pipeline (AWT) 647
Renjin project 387, 388
repaint method
of Component 451, 453
replaceAll method
of Matcher 145, 147
of String 145
replaceFirst method
of Matcher 146, 147
#REQUIRED 167
required option (JAAS configuration) 410
requisite option (JAAS configuration) 410
RescaleOp class 703, 709
Constructor 709
Rescaling operation 703
reset method
of Buffer 108, 109
of InputStream 59
of MessageDigest 421
resetChoosableFileFilters method
of JFileChooser 578
resetChoosableFilters method 575
resolve method
of Path 87, 88
resolveEntity method 164
of EntityResolver 169
resolveSibling method 87
of Path 88
Resolving
classes 398
relative URLs 233
Resource bundles 369
loading 371
locating 369
lookup tables for 371
naming 371
searching for 371
Resource editors 369
ResourceBundle class 373
extending 371, 372
getBundle method 370, 373
getKeys method 372, 373
getObject method 371, 372, 373
getString method 371, 373
getStringArray method 373
handleGetObject method 372, 373
ResourceBundle.Control class
getCandidateLocales method 370
Resources
hierarchy of 370
Response headers 235, 236
Response pages 239
Result interface 204, 205, 314
Result sets
accessing columns in 277
analyzing 276
closing 279
for multiple queries 278
iterating over rows in 294
metadata for 304
numbering rows in 296
order of rows in 277
retrieving multiple 293
scrollable 294
updatable 294, 296
results method
of Matcher 147
ResultSet interface 278, 291, 299, 300, 309
absolute method 296, 299
afterLast method 299
afterLast, beforeFirst methods 296
beforeFirst method 299
cancelRowUpdates method 297, 299
close method 278, 279
concurrency values 297
deleteRow method 298, 299
findColumn method 278
first method 296, 299
getBlob method 291
getBlob, getClob methods 290
getClob method 291
getConcurrency method 295, 297, 299
getDate, getDouble, getInt methods 277
getMetaData method 304, 309
getObject method 277, 278
getRow method 296, 299
getString method 277
getType method 295, 299
get*Xxx* method 278
insertRow method 297, 299
isAfterLast method 299
isAfterLast, isBeforeFirst methods 296
isBeforeFirst method 299
isClosed method 278
isFirst method 299
isFirst, isLast methods 296
isLast method 299
iteration protocol 276
last method 296, 299
moveToCurrentRow method 299
moveToInsertRow method 299
moveTo*Xxx*Row methods 297
next method 276, 278, 294
previous method 295, 296, 299
relative method 296, 299
type values 295, 297
updateObject method 278
updateRow method 299
update*Xxx* method 299
update*Xxx* methods 297

ResultSetMetaData interface 304, 309
  getColumnCount method 309
  getColumnDisplaySize method 309
  getColumnLabel method 309
  getColumnName method 309
  getColumn*Xxx* methods 304
Retirement calculator example 373
Return values, missing 13
revalidate method
  of JComponent 508, 509
rewind method
  of Buffer 108, 109
RFC 1123 332
RFC 2396 232
RFC 2616 235
RFC 2911 738
RFC 821 259
RFC 822 332
RGB 680, 698
Rhino 388, 394
Rivest, Ronald 420, 423, 438
Role-based authentication 412
rollback method
  of Connection 310, 312
Root component 86
Root element 152
  referencing schemas in 170
Root node 614, 620
  handles for 620
  separating children of 618
rotate method
  of Graphics2D 674, 675, 678
Rotation 673, 674
  interpolating pixels and 703, 704
  with center point 675
Round cap 665
Round join 665
Rounded rectangles 651
RoundRectangle2D class 650, 651
RoundRectangle2D.Double class 651, 662
  Constructor 662
RoundRectangle2D.Float class 651
Row sets 300
  cached 300, 305
  constructing 300
  modifying 301
  page size of 301
RowFilter class 594, 595, 601
  andFilter method 595, 602
  dateFilter method 594, 602
  include method 595, 601
  notFilter method 595, 602
  numberFilter method 594, 602
  orFilter method 595, 602
  regexFilter method 594, 602
RowFilter.Entry class 595, 602
  getIdentifier method 602
  getModel method 602
  getStringValue method 602
  getValue method 602
  getValueCount method 602
ROWID 313, 314
RowId interface 314
Rows (databases) 266
  deleting/inserting 298
  iterating through 297
  order of, in result set 277
  retrieving 314
  selecting 269
  updating 296
Rows (Swing)
  filtering 594, 595
  height of 591
  hiding 595
  margins of 591
  position, in a node 624
  resizing 591
  selecting 583, 592
  sorting 583, 592, 594
RowSet interface 300, 302
  execute method 302
  getCommand method 302
  getPassword method 302
  getURL method 302
  getUsername method 302
  setCommand method 302
  setPassword method 302
  setURL method 302
  setUsername method 302
  set*Xxx* methods 301
RowSetFactory interface 303
  createCachedRowSet method 300, 301, 303
  createFilteredRowSet method 303
  createJdbcRowSet method 303
  createJoinRowSet method 303
  createWebRowSet method 303
RowSetProvider class 303
  newFactory method 300, 303
RSA algorithm 423, 438
RSA Security, Inc. 432
rt.jar file 399
Ruby 387
run method
  of Tool 379, 385
Runtime class
  exec method 132

S

S (short), type code 119
Sample values of pixels 697
Sandbox 430
Save points 311
Savepoint interface 312
  getSavepointId method 312
  getSavepointName method 312
SAX (Simple API for XML) parser 154, 185
  activating namespace processing in 187
SAXParseException class 170
  getColumnNumber method 170
  getLineNumber method 170
SAXParser class 188
  parse method 186, 188
SAXParserFactory class 188
  isNamespaceAware method 188
  isValidating method 188
  newInstance method 188
  newInstance, newSAXParser methods 186

newNSInstance method 188
newSAXParser method 188
setFeature method 187
setNamespaceAware method 188
setValidating method 188
SAXResult class 205
SAXSource class 204, 205, 210
Constructor 210
Scalar functions 291, 292
Scale 673
scale method
of Graphics2D 674, 678
Scaling 673, 674
Scanner class 9, 68, 148
constructor 88, 224
findAll method 148
tokens method 5, 9
useLocale method 345, 347
Scheduling applications
computing dates for 325
time zones and 321, 327
schema element (XML Schema) 173
Scheme 387
ScriptContext interface 390, 391
getErrorWriter method 391
getReader method 391
getWriter method 391
get*Xxx*/set*Xxx* methods 391
setErrorWriter method 391
setReader method 391
setWriter method 391
ScriptEngine interface 390, 391
createBindings method 390
eval method 389, 390
get method 390
getContext method 391
put method 390
ScriptEngineFactory interface 389
getExtensions method 389
getMethodCallSyntax method 392
getMimeTypes method 389
getNames method 389
ScriptEngineManager class 388, 390
get method 390
getEngineByExtension method 388
getEngineByMimeType method 388
getEngineByName method 388
getEngineFactories method 388
getEngine*Xxx* methods 388
put method 390
Scripting engines 388
adding variable bindings to 389
calling functions in 391
implementing Java interfaces 392
invoking 388
Scripting languages 387
Scripts 387
accessing classes in 393
compiling 393
executing 389
invoking 389
redirecting I/O of 391
using Java method call syntax in 392
Scroll panes 511
with tables 582
with trees 625, 626
Scrollbars 512
scrollPathToVisible method 625
of JTree 629
SecretKey interface 433
SecretKeySpec class 437
Constructor 437
Secure random generator 434
SecureRandom class
setSeed method 434
Security class 397, 440
bypassing constructors 132
bytecode verification 405
class loaders 398
code signing 397
different levels of 418
digital signatures 418
encryption 430
user authentication 409
Security manager 397
"Seek forward only" mode (ImageInputStream) 690
seek method
of RandomAccessFile 80, 84
SegmentAllocator interface 750
allocate method 750
allocateFrom method 750
elements method 750
SELECT 269
executing 276
for LOBs 290
multiple, in a query 293
not supported in batch updates 311
select attribute (XSLT) 204
Selection models 592
Semantic events 488
send method
of HttpClient 253
sendAsync method 249
of HttpClient 253
separator constant of File class 64
Separators 64, 86
sequence element (XML Schema) 172
sequenceElement method
of MemoryLayout.PathElement 753, 765
SequenceLayout interface 752
sequenceLayout method
of MemoryLayout 753
Sequences, producing 4
Serial numbers 115
SerialCloneable 133
Serializable interface 114, 119
readResolve, writeReplace methods 128
Serialization 113
cloning with 133
file format for 118
filters for 133
serial numbers for 115
serialver program 129
serialVersionUID method 129, 131
Server-side programs 239
redirecting URLs in 244
Servers
accessing 231

connecting clients to 213
echo 220
implementing 218
ServerSocket class 218, 221
accept method 218, 221
close method 221
Constructor 221
ServerSocketFactory class 230
createServerSocket method 230
Service provider interfaces 689
Servlets 383
Session class
setDebug method 261
set method
of MemorySegment 751
setAccelerator method 537
of JMenuItem 538
setAcceptAllFileFilterUsed method 575
of JFileChooser 578
setAccessory method
of JFileChooser 578
setAction method
of AbstractButton 533
setActionCommand method
of AbstractButton 519
setAllowsChildren method 621
of DefaultMutableTreeNode 623
setAllowUserInteraction method 233
setAsksAllowsChildren method 621
of DefaultTreeModel 622
setAtIndex method
of MemorySegment 751
setAttribute method
of Element 195
setAttribute, setAttributeNS methods
of Element 193
setAttributeNS method
of Element 195
setAutoCommit method 310, 312
of Connection 312
setAutoCreateRowSorter method 583, 592
of JTable 585
setAutoResizeMode method 591
of JTable 600
setBackground method 461
of Component 461
setBinaryStream method
of Blob 291
setBorder method 520
of JComponent 521
setBounds method 447
of Component 449
setCellEditor method 606
of TableColumn 612
setCellRenderer method
of TableColumn 612
setCellSelectionEnabled method 592
of JTable 600
setCharacterStream method
of Clob 291
setClip method 678, 713
of Graphics 680
setClosedIcon method 632
of DefaultTreeCellRenderer 633
setColumns method
of JTextArea 511, 513
of JTextField 507, 508
setColumnSelectionAllowed method 592
of JTable 600
setCommand method 301
of RowSet 302
setComparator method 593
of DefaultRowSorter 601
setComponentPopupMenu method 536
of JComponent 536
setComposite method 648, 682
of Graphics2D 687
setConnectTimeout method 233
of URLConnection 238
setContentHandler method
of XMLReader 210
setContextClassLoader method 400
of Thread 405
setCrc method
of ZipEntry 86
setCurrency method 352
setCurrentDirectory method 572
of JFileChooser 577
setCursor method
of Component 486
setDataElements method 699
of WritableRaster 702
setDate method 285
setDebug method 261
setDecomposition method
of Collator 361
setDefault method
of CookieHandler 244
of Locale 343, 345
setDefaultButton method 568
of JRootPane 572
setDefaultCloseOperation method 565
setDefaultNamespace method
of XMLStreamWriter 200
setDefaultRenderer method 604
setDisplayedMnemonicIndex method 537
of AbstractButton 538
setDoInput method 233, 234
of URLConnection 238
setDoOutput method 233, 234, 241, 243
of URLConnection 238
setDouble method 285
setEchoChar method
of JPasswordField 510
setEditable method 626
of JComboBox 522, 524
of JTextComponent 507
setEnabled method
of Action 477, 481
of JMenuItem 539, 540
setEntityResolver method 164
of DocumentBuilder 169
setErrorHandler method
of DocumentBuilder 169
setErrorWriter method
of ScriptContext 391
setFeature method 187
setFileFilter method 575

of JFileChooser 578
setFileSelectionMode method 573
of JFileChooser 578
setFileView method 575, 576
of JFileChooser 578
setFillsViewportHeight method
of JTable 585
setFont method
of JComponent 509
setForeground method 461
of Component 461
setFrameFromCenter method 456
setFrom method 260
setGroupingUsed method
of NumberFormat 349
setHeaderRenderer method
of TableColumn 612
setHeaderValue method
of TableColumn 612
setHeader*Xxx* methods
of TableColumn 604
setHorizontalTextPosition method 533
of AbstractButton 534
setIcon method
of JLabel 510
of JMenuItem 533
setIconImage method 447
of Frame 449
setIfModifiedSince method 233, 235
of URLConnection 238
setIgnoringElementContentWhitespace method 168
of DocumentBuilderFactory 170
setInheritsPopupMenu method 536
of JComponent 536
setInput method
of ImageReader 695
setInstanceFollowRedirects method 244
setInt method 285
setInverted method 527
setJMenuBar method 530
of JFrame 533
setLabelTable method 526
of JSlider 530
setLayout method
of Container 503
setLeafIcon method 632
of DefaultTreeCellRenderer 633
setLevel method
of ZipOutputStream 85
setLineWrap method 511
of JTextArea 513
setLocale method
of MessageFormat 363
setLocation method 447
of Component 449
setLogWriter method 274
setMajorTickSpacing method
of JSlider 530
setMaximumFractionDigits method
of NumberFormat 349
setMaximumIntegerDigits method
of NumberFormat 349
setMaxWidth method 590
of TableColumn 601
setMethod method
of ZipEntry 86
of ZipOutputStream 86
setMinimumFractionDigits method
of NumberFormat 349
setMinimumIntegerDigits method
of NumberFormat 349
setMinorTickSpacing method
of JSlider 530
setMinWidth method 590
of TableColumn 601
setMnemonic method 537
of AbstractButton 538
setModel method 523
setMultiSelectionEnabled method 573
of JFileChooser 577
setName method
of NameCallback 417
setNamespaceAware method
of DocumentBuilderFactory 173, 183, 185, 187, 193
of SAXParserFactory 188
setOpenIcon method 632
of DefaultTreeCellRenderer 633
setOutput method
of ImageWriter 696
setOutputProperty method 194
of Transformer 195
setPageable method 719
setPageSize method 301
of CachedRowSet 303
setPaint method 460, 648, 671
of Graphics2D 461, 673
setPaintLabels method 526
of JSlider 530
setPaintTicks method 526, 527
of JSlider 530
setPaintTrack method
of JSlider 530
setParseIntegerOnly method
of NumberFormat 349
setPassword method
of PasswordCallback 418
of RowSet 301, 302
setPixel method
of WritableRaster 702
setPixel, setPixels methods
of WritableRaster 697
setPixels method
of WritableRaster 702
setPreferredWidth method 590
of TableColumn 601
setPrefix method
of XMLStreamWriter 200
setPrintable method
of PrinterJob 718
setProperty method 190
of XMLInputFactory 191
setReader method 391
of ScriptContext 391
setReadTimeout method 233
of URLConnection 238
setRenderingHint, setRenderingHints methods
of Graphics2D 648
setRequestProperty method 233, 235

of URLConnection 238
setResizable method 447, 590
of Frame 449
of TableColumn 601
setRootVisible method 620
of JTree 622
setRowFilter method 594, 595
of DefaultRowSorter 601
setRowHeight method
of JTable 600
setRowHeight, setRowMargin methods
of JTable 591
setRowMargin method
of JTable 600
setRows method 511
of JTextArea 513
setRowSelectionAllowed method 592
of JTable 600
setRowSorter method 592
of JTable 600
setSavepoint method
of Connection 312
setSeed method 434
setSelected method
of AbstractButton 535
of JCheckBox 514, 516
setSelectedFile method
of JFileChooser 577
setSelectedFile/Files methods
of JFileChooser 573
setSelectedFiles method
of JFileChooser 577
setSelectionMode method 592
of ListSelectionModel 601
setShowsRootHandles method 620
of JTree 622
setSize method
of Component 449
of ZipEntry 86
setSnapToTicks method 526
of JSlider 530
setSortable method 593
of DefaultRowSorter 601
setSoTimeout method 215
of Socket 216
setStrength method
of Collator 361
setString method
of PreparedStatement 285
of RowSet 301
setStringConverter method
of TableRowSorter 601
setStroke method 648, 664
of Graphics2D 671
setSubject method 260
setTableName method 302
of CachedRowSet 303
setTabSize method
of JTextArea 513
setText method 260
of JLabel 510
of JTextComponent 507, 508
setTitle method 447
of Frame 449
setToolTipText method
of JComponent 546
setToRotation method
of AffineTransform 678
setToScale method
of AffineTransform 678
setToShear method
of AffineTransform 678
setToTranslation method
of AffineTransform 678
setTo*Xxx* methods
of AffineTransform 676
setTransform method 676
of Graphics2D 678
setURL method 301
of RowSet 302
setUseCaches method 233
setUsername method 301
of RowSet 302
setUserObject method 616
of MutableTreeNode 622
setValidating method
of DocumentBuilderFactory 168, 170
of SAXParserFactory 188
setValueAt method 608
of TableModel 588
setVisible method
of Component 447, 449
of JDialog 564, 566, 567
setWidth method 591
of TableColumn 601
setWrapStyleWord method
of JTextArea 513
setWriter method 391
of ScriptContext 391
set*Xxx* method
of PreparedStatement 289
SGML 150
SHA-1 118, 419
Shamir, Adi 423, 438
Shape interface 454, 650
implementing 664
ShapeMaker
getPointCount method 656
makeShape method 656
ShapePanel 656
Shapes
clipping 647, 648, 678
combining 648, 663
control points of 656
drawing 647, 650
filling 647, 649, 671
rendering 649
transforming 647
Shared arena 748
Shared libraries 747
Shear 673, 674
shear method
of Graphics2D 674, 678
Shift-JIS 75
short type
printing 70
streams of 46
type code for 119

writing in binary format 78
ShortBuffer class 107
ShortLookupTable class 704, 710
Constructor 710
shouldSelectCell method 607, 608
of CellEditor 613
show method
of JPopupMenu 535, 536
showConfirmDialog method 559
of JOptionPane 562
showDialog method 568, 572, 574
of JFileChooser 578
showInputDialog method 559, 560
of JOptionPane 563
showInternalConfirmDialog method
of JOptionPane 562
showInternalInputDialog method
of JOptionPane 563
showInternalMessageDialog method
of JOptionPane 562
showInternalOptionDialog method
of JOptionPane 563
showMessageDialog method 559
of JOptionPane 562
showOpenDialog method 572, 573
of JFileChooser 578
showOptionDialog method 559, 561
of JOptionPane 563
showSaveDialog method 572, 573
of JFileChooser 578
shutdownInput method
of Socket 224
shutdownOutput method
of Socket 224
Signatures. See Digital signatures
Simple types 171
SimpleDateFormat class 362
SimpleDoc class 728, 729
Constructor 729
SimpleFileVisitor class 99, 100
postVisitDirectory method 100
preVisitDirectory method 100
visitFile method 99, 100
visitFileFailed method 99, 100
SimpleJavaFileObject class 381, 387
getCharContent method 387
openOutputStream method 387
SimpleScriptContext class 390
simpleType method 171
Singletons 128
size method
of BasicFileAttributes 95
of Files 93, 94
skip method
of InputStream 59
of Stream 11, 12
skipBytes method
of DataInput 80
skipNBytes method
of InputStream 59
Sliders 525
ticks on 526, 527
vertical 525
SMALLINT 271, 313
SMTP 259
Socket class 215, 216, 224
connect method 216
Constructor 215, 216
getInputStream method 215, 218
getOutputStream method 215, 218
isClosed method 216
isConnected method 216
isInputShutdown method 224
isOutputShutdown method 224
setSoTimeout method 215, 216
shutdownInput method 224
shutdownOutput method 224
SocketChannel class 224, 227
open method 224, 227
SocketFactory class 231
createSocket method 231
Sockets
half-closing 223
interrupting 224
opening 214
server 218
timeouts of 215
SocketTimeoutException class 215
SoftBevelBorder class 520, 521
Constructor 521
Software developer certificates 430
sort method
of Collections 357
sorted method
of Stream 12, 13
Source files
character encoding of 368
reading from memory 381
Source interface 204, 314
Space. See Whitespace
SPARC processor, big-endian order in 78
split method
of Pattern 144, 147
of String 8, 72, 144
splitAsStream method 4, 144
of Pattern 8, 147
spliterator method
of Iterable 8
Spliterators class 8
spliteratorUnknownSize method 5, 8
spliteratorUnknownSize method
of Spliterators 8
splitWithDelimiters method
of Pattern 147
Spring layout 546
SQL 263, 265
changing data inside databases 270
commands in 273
data types in 271, 313
equality testing in 269
escapes in 291
exceptions in 279
executing statements in 276
keywords in 268
reading instructions from a file 281
strings in 269
vs. Java 286
warnings in 280

wildcards in 269
SQLException class 279, 280, 295
getErrorCode method 281
getNextException method 280
getSQLState method 281
getXxx methods 279
iterator method 279, 280
rolling back and 311
SQLWarning class 280, 281, 295
getNextWarning method 281
SQLXML interface 313, 314
Square cap 665
Square root, computing 18
SQuirreL 305
SRC, SRC_Xxx composition rules 682, 683
SRGB standard 698
SSLServerSocketFactory class 230
getDefault method 230
SSLSocketFactory class 230
getDefault method 230
Standard extensions 399
Standard input 58
StandardCharsets class 75
StandardJavaFileManager interface 380, 381, 386
getJavaFileObjectsFromFiles method 386
getJavaFileObjectsFromPaths method 386
getJavaFileObjectsFromStrings method 386
start method
of MatchResult 148
startDocument method 185
of ContentHandler 188
startElement method 185
of ContentHandler 189
stateChanged method 525
Stateless operations 51
Statement class 276
addBatch method 311
close, closeOnCompletion methods 279
execute method 282, 293
executeBatch method 312
executeQuery method 276, 295, 297
executeUpdate method 276, 310, 311
getMoreResults method 293
getUpdateCount method 293
using for multiple queries 278
Statement interface 277, 294, 313
addBatch method 313
close method 278
closeOnCompletion method 278
execute method 278, 294
executeBatch method 313
executeLargeBatch method 313
executeLargeUpdate method 277
executeQuery method 277
executeUpdate method 277, 294
getLargeUpdateCount method 278
getMoreResults method 294
getResultSet method 278
getUpdateCount method 278
isClosed method 278
Statements
closing 279
complex 286
concurrently open 279
executing 276
grouping into transactions 310
in batch updates 311
multiple 278
prepared 284
truncations in 280
Static initialization blocks 745
Static inner classes 121
statusCode method 249
of HttpResponse 254
StAX parser 189, 196
namespace processing in 190
no indented output in 197
StAXSource 204
stopCellEditing method 607, 608
of CellEditor 613
Stored procedures 291, 292
Stream 40, 45
collect method 40
gather method 45
reduce method 40
Stream interface 3, 7, 10, 12, 13, 14, 22
allMatch method 14
allMatch, anyMatch methods 14
anyMatch method 14
collect method 22, 25, 39
concat method 11, 12
count method 2, 3, 13
distinct method 12, 13, 52
dropWhile method 11, 12
empty method 4, 7
filter method 2, 3, 9, 10, 13
findAny method 14
findFirst method 13, 14
flatMap method 10, 11
forEach method 22
forEachOrdered method 22
generate method 4, 7, 47
iterate method 4, 7, 12, 47
iterator method 21
limit method 11, 12, 52
map method 9, 10
mapMulti method 11
mapToInt method 39
max method 14
max, min methods 13
min method 14
noneMatch method 14
of method 3, 7
ofNullable method 4, 7, 20
peek method 12, 13
reduce method 38
skip method 11, 12
sorted method 12, 13
takeWhile method 11, 12
toArray method 22
toList method 5, 22
unordered method 52
stream method
of Arrays 3, 8, 47
of Collection 2, 3
of Optional 19
of StreamSupport 5, 8
Streaming parsers 154, 185

StreamPrintService class 729
StreamPrintServiceFactory class 729
  getPrintService method 729
  lookupStreamPrintServiceFactories method 729
StreamResult class 196, 205
  Constructor 196
Streams class 1
  collecting results from 21
  computing values from 38
  converting to/from arrays 3, 22, 52
  creating 3
  debugging 12
  empty 4, 13, 38, 39
  encrypted 437
  filtering 20
  finite 4
  flattening 10, 19
  for print services 729
  infinite 2, 4, 11, 12
  intermediate operations for 2
  noninterference of 6
  of primitive types 39
  of random numbers 47
  parallel 2, 14, 22, 28, 30, 39, 51
  processed lazily 2, 6, 12
  reductions of 13
  removing duplicates from 12
  returned by Files.lines 53
  sorted 12, 52
  splitting/combining 11
  summarizing 23
  terminal operations for 2, 13
  transformations of 9, 47
  vs. collections 2
StreamSource class 204, 209
  Constructor 209
  transform method 205
StreamSupport class 8
  stream method 5, 8
String class 8, 63
  compareTo method 356
  format method 345
  lines method 8
  replaceAll method 145
  split method 8, 72, 144
  strip method 157, 347, 508
  toLowerCase method 9, 344
  toUpperCase method 344
String parameters 744
StringBuffer class 63, 108
StringBuilder class 63, 81
Strings
  converting to grapheme clusters 9
  encoding 339, 368
  fixed-size, I/O of 81, 82
  in native code 744
  in SQL 269
  internationalizing 369
  ordering 356
  patterns for 134
  printing 70
  sorting 357
  splitting 4
  transforming to lower/uppercase 9, 344
  writing in binary format 78
StringSource 381
strip method
  of String 157, 347, 508
Stroke interface 664
StrokePanel 666
Strokes 647, 664
  dashed 666
  setting 648
  styling 665, 666
StructLayout interface 752
structLayout method
  of MemoryLayout 753
Stylesheets 201
Subject class 412
  getPrincipals method 412
Subjects (logins) 410
Submenus 530, 531
subSequence method
  of CharSequence 64
subtract method
  of Area 664
Subtraction operator, not associative 39
Subtrees 617, 631
  adding nodes to 625
  collapsed and expanded 617, 618
Suetonius, Gaius Tranquillus 402
sufficient option (JAAS configuration) 410
sum method
  of DoubleStream 50
  of IntStream 49
  of LongStream 49
sum, summaryStatistics methods (primitive streams) 48
summarizingDouble method
  of Collectors 24
summarizingInt method
  of Collectors 24
summarizingLong method
  of Collectors 24
summarizing*Xxx* methods
  of Collectors 23, 32
summaryStatistics method
  of DoubleStream 50
  of IntStream 49
  of LongStream 49
summingDouble method
  of Collectors 35
summingInt method
  of Collectors 35
summingLong method
  of Collectors 35
summing*Xxx* methods
  of Collectors 31
Sun 263
Sun Microsystems 444
SunJCE ciphers 432
Superclasses
  adding 131
  not serializable 116
Supplier interface 9
  get method 9
supplier method
  of Collector 38
SupportedValuesAttribute interface 731

supportsBatchUpdates method 311
of DatabaseMetaData 313
supportsResultSetConcurrency method
of DatabaseMetaData 300
supportsResultSetType method
of DatabaseMetaData 300
supportsResultSet*Xxx* methods (DatabaseMetaData) 295
SVG 197
Swing 443, 581
building GUI with 497
generating dynamic code for 383
model-view-controller analysis of 499, 501
starting 446
tables in 581
trees in 613
SwingConstants interface 509
SwingUtilities class 571
getAncestorOfClass method 571
SwingUtilities class, getAncestorOfClass method 568
SymbolLookup interface 754, 759
find method 759
findOrThrow method 759
libraryLookup method 759
Symmetric ciphers 431
performance of 438
SyncProviderException 302
SYSTEM 163, 194
System class 747
console method 367
loadLibrary method 745, 747
System class loader 399
System.err 70, 379
System.in 58, 70
character encoding and 367
System.out 70, 379
character encoding and 367
println method 367
systemNodeForPackage method
of Preferences 494
systemNodeForPackage, systemRoot methods of Preferences 490
systemRoot method
of Preferences 494

T

t (SQL escape) 292
Table cell renderers 589
Table models 581, 585
updating after cell editing 608
TableCellEditor interface 612
getTableCellEditorComponent method 607, 608, 612
implementing 607, 608
TableCellRenderer interface 612
getTableCellRendererComponent method 602, 612
implementing 602
TableColumn class 590, 591, 596, 601, 612
Constructor 601
setCellEditor method 606, 612
setCellRenderer method 612
setHeaderRenderer method 612
setHeaderValue method 612
setHeader*Xxx* methods 604
setMaxWidth method 601
setMinWidth method 601
setPreferredWidth method 601
setResizable method 601
setResizable, set*Xxx*Width methods 590
setWidth method 591, 601
TableColumnModel interface 590, 600
getColumn method 590, 600
TableModel interface 588, 592, 599
getColumnClass method 588, 599
getColumnCount method 588
getColumnName method 588
getColumnType method 605
getRowCount method 588
getValueAt method 585, 586, 588, 605
get*Xxx*Count methods 585, 586
implementing 585
isCellEditable method 588, 604
setValueAt method 588, 608
TableRowSorter class 592, 594, 601
setStringConverter method 601
Tables (databases) 266
changing data in 270
creating 270
duplicating data in 267
inspecting 268
metadata for 304
multiple, selecting data from 269
removing 274
Tables (Swing) 581
asymmetric 588
cells in
editing 604, 606
rendering 602, 604
selecting 592
columns in
accessing 590
adding 596
hiding 596
naming 586
rearranging 583
rendering 588, 590
resizing 583, 590, 591
selecting 592
combo boxes in 605
constructing 582, 586
custom editors in 606, 608
headers in 582
rendering 604
index values in 592
printing 584
relationship between classes of 590
rows in
filtering 594, 595
hiding 595
margins of 591
resizing 591
selecting 583, 592
sorting 583, 592, 594
scrolling 582
text fields in 605
TableStringConverter class 601
toString method 594, 601
takeWhile method 11
of Stream 12

TCP 215
teeing method
of Stream 33
Telnet 211
activating/connecting 211
windows communicating in 221, 222
template element (XSLT) 203
TemporalAdjuster interface 325
TemporalAdjusters class 325, 326
dayOfWeekInMonth method 326
firstDayOfMonth method 326
firstDayOfNextMonth method 326
firstDayOfNextYear method 326
firstDayOfYear method 326
lastDayOfMonth method 326
lastDayOfYear method 326
lastInMonth method 326
next method 326
nextOrSame method 325, 326
ofDateAdjuster method 325
previous method 326
previousOrSame method 326
Text areas 511
formatted text in 512
preferred size of 511
scrollbars in 512
Text fields 507, 605
columns in 507
creating blank 508
preferred size of 507
Text files, encoding of 366, 368
Text input 507
labels for 509
password fields 510
scroll panes 511
Text interface 67
centering 463
displaying 452
fonts for 461
generating from XML files 204, 206
output 69
printing 710, 726
reading 68
saving objects in 71
transmitting through sockets 218
typesetting properties of 463
vs. binary data 67
Text nodes
constructing 193
retrieving from XML 157
TextCallbackHandler class 413
TextLayout class 679, 680
Constructor 680
getAdvance method 680
getAscent method 680
getDescent method 680
getLeading method 680
TextStyle enumeration 354
TextSyntax class 734
TexturePaint class 671, 673
Constructor 673
Thread class 405
get/setContextClassLoader methods 400
getContextClassLoader method 405
setContextClassLoader method 405
Threads
blocking 58, 224
executing scripts in 389
Internet connections with 221
race conditions in 52
referencing class loaders in 400
Thumbnails 690
Ticks 526
icons for 527
labeling 526
snapping to 526
TIME 271, 292, 313
current 318
formatting 331, 353
instances of 326
literals for 291
local 326
measuring 318
parsing 334
zoned 327, 353
Time class 336
valueOf method 337
Time of day service 212
Timeouts 215
Timer class 488
TIMESTAMP 271, 292, 313
Timestamp class 336
toInstant method 337
valueOf method 337
Timestamps 332
using instants as 318
TimeZone class
getTimeZone method 337
toZoneId method 337
toAbsolutePath method 88
of Path 89
toArray method
of AttributeSet 738
of DoubleStream 49
of IntStream 49
of LongStream 49
of MemorySegment 751
of primitive streams 47
of Stream 22
toCollection method 25
of Collectors 27
toConcurrentMap method 28
of Collectors 29
toDays method 318
toDaysPart method
of Duration 320
toFile method 88
of Path 89
toFormat method 333, 337
toHours method 318
toInstant method
of Date 336
of FileTime 95, 337
of Timestamp 337
of ZonedDateTime 328, 331
tokens method
of Scanner 5, 9
toLanguageTag method 342

of Locale 345
toList method
of Collectors 25, 26
of Stream 5, 22
toLocalDate method
of ZonedDateTime 331
toLocalDateTime method
of ZonedDateTime 331
toLocalTime method
of ZonedDateTime 331
toLocal*Xxx* methods
of Local*Xxx* 337
toLowerCase method 9, 344
toMap method 27
of Collectors 29
toMillis, toMinutes, toNanos methods
of Duration 318
toNanoOfDay method
of LocalTime 327
Tool interface 385
run method 379, 385
Toolbars 543
detaching 543
dragging 543
title of 545
vertical 545
Toolkit class 449
getDefaultToolkit method 448, 449
getScreenSize method 448, 449
ToolProvider class
getSystemJavaCompiler method 379
Tooltips 545
toPath method 88
of File 89
toSecondOfDay method
of LocalTime 327
toSeconds method 318
of Duration 320
toSet method 25, 31
of Collectors 26
toString method
of CharSequence 64
of Currency 353
of Locale 345
of TableStringConverter 594, 601
of Variable 642
toUnmodifiableList method
of Collectors 26
toUnmodifiableMap method
of Collectors 29
toUnmodifiableSet method
of Collectors 26
toUpperCase method 344
to*Xxx* method
of Duration 320
to*Xxx*Part method
of Duration 320
toZonedDateTime method 336
toZoneId method 337
Transactions 310
committing 310
error handling in 312
rolling back 310
transferTo method
of InputStream 59
transform method
of Graphics2D 648, 676, 678
of StreamSource 205
of Transformer 194, 195, 204
Transformations 647, 673
affine 676, 703
composing 674, 675
fundamental types of 673, 674
matrices for 675, 676
order of 674
setting 648
using for printing 720
Transformer class 195
setOutputProperty method 194, 195
transform method 194, 195, 204
TransformerFactory class 195, 209
newInstance method 194, 195
newTransformer method 194, 195, 209
transient keyword 125
Translate 720
translate method
of Graphics2D 674, 678
translatePoint method 489
Translation 673, 674
Transparency interface 680
Traversal order 629, 630
Tree events 633
Tree models
constructing 615, 640
custom 639
default 615
Tree parsers 154
Tree paths 623, 626
constructing 625, 630
Tree selection listeners 634
TreeCellRenderer interface 631, 633
getTreeCellRendererComponent method 632, 633
implementing 631
TreeMap class 28
TreeModel interface 614, 622, 623, 646
add/removeTreeModelListener methods 641
addTreeModelListener method 647
getChild method 646
getChild, getChildCount methods 640, 642
getChildCount method 646
getIndexOfChild method 640, 646
getRoot method 640, 642, 646
implementing 615
isLeaf method 622, 641, 646
removeTreeModelListener method 647
valueForPathChanged method 641, 647
TreeModelEvent class 647
Constructor 647
TreeModelListener interface 641, 647
treeNodesChanged method 647
treeNodesInserted method 647
treeNodesRemoved method 647
treeNodes*Xxx* methods 641
treeStructureChanged method 641, 647
TreeNode interface 615, 622, 623, 629
children method 629
getAllowsChildren method 622
getChildAt method 629

getChildCount method 629
getParent method 629, 630
isLeaf method 621, 622
treeNodesChanged method
of TreeModelListener 647
treeNodesInserted method
of TreeModelListener 647
treeNodesRemoved method
of TreeModelListener 647
TreePath class 623, 629
getLastPathComponent method 623, 629
Trees 613
adding listeners to 634
background color for 632
connecting lines in 618, 619
displaying 614
editing 623, 626, 641
handles in 617, 620, 631
hierarchy of classes for 615
indexes in 624
infinite 643
leaves in 614, 621, 631, 640
nodes in 614, 621, 631, 640
paired with other components 633
rendering 631
scrolling to newly added nodes 625, 626
structure of 614
subtrees in 617, 618
traversals for 629
updating vs. reloading 625
user objects for 615, 625
view of 624, 625
with horizontal lines 618
TreeSelectionEvent class 639
getPath method 639
getPaths method 635, 639
TreeSelectionListener interface 639
implementing 633
valueChanged method 634, 635, 639
TreeSelectionModel interface 634
treeStructureChanged method
of TreeModelListener 647
Troubleshooting. See Debugging
True Odds: How Risks Affect Your Everyday Life (Walsh) 419
truncate method
of FileChannel 113
try-with-resources statement 62
closing files with 95, 98
for database connections 279
with locks 110
tryLock method 110
of FileChannel 113
ts (SQL escape) 292
Type codes 119
Type definitions 171
anonymous 172
nesting 172
type method
of XPathEvaluationResult 182
TYPE_BICUBIC, TYPE_BILINEAR fields
of AffineTransformOp 703
TYPE_BYTE_GRAY 699
TYPE_INT_ARGB 697, 698
TYPE_NEAREST_NEIGHBOR field
of AffineTransformOp 703
Types. See Data types
Typesetting terms 463

U

U.S. government on exporting encryption methods 402
UDP 215
UIManager class 603
UncheckedIOException class 68
Unicode 47, 74
character order in 356
converting to binary data 67
in property files 371
input/output streams and 57
native code and 744
normalization forms in 358
using for all strings 339
unionLayout method
of MemoryLayout 753
Units of measurement 153
UNIX
authentication in 409
line feed in 70, 367
paths in 87
specifying locales in 343
UnixLoginModule 410
UnixNumericGroupPrincipal class 410
UnixPrincipal class 410
UnknownHostException class 214
unordered method
of BaseStream 55
of Stream 52
Unparsed external entities 167
unread method
of PushbackInputStream 67
UnsatisfiedLinkError class 742
until method
of LocalDate 322, 324
upcallStub method
of Linker 761
UPDATE 270, 285, 297
executing 276
in batch updates 311
truncations in 280
vs. methods of ResultSet 298
update method
of Cipher 432, 434, 436, 437
of MessageDigest 420, 421
updateObject method
of ResultSet 278
updateRow method
of ResultSet 299
update*Xxx* method
of ResultSet 299
update*Xxx* methods (ResultSet) 297
URI class 248
get*Xxx* methods 232
no resource accessing with 231
uri method
of HttpRequest.Builder 247, 249, 254
URIs 182, 231
absolute vs. relative 232, 233

base 233
hierarchical 232
namespace 182
opaque vs. nonopaque 232
schemes for 232
with HTTP 248
URISyntax class 734
URL class 231, 238, 248
accepted schemes for 231
openConnection method 233, 238
openStream method 64, 231, 238
URLClassLoader class 405
addURLs method 399
Constructor 405
getURLs method 399
loadClass method 399
URLConnection class 231, 233, 238, 247
connect method 234, 235, 238
getConnectTimeout method 238
getContent method 239
getContentEncoding method 239
getContentEncoding, getContentType methods 234, 236, 243
getContentLength method 234, 236, 239
getContentType method 239
getDate method 234, 236, 239
getDoInput method 238
getDoOutput method 238
getExpiration method 234, 236, 239
getHeaderField method 238
getHeaderFieldKey method 238
getHeaderFields method 238
getHeader*Xxx* methods 234, 236
getIfModifiedSince method 238
getInputStream method 234, 239, 242, 243
getLastModifed method 239
getLastModified method 234, 236
getOutputStream method 234, 239, 241
getReadTimeout method 238
getRequestProperties method 238
setAllowUserInteraction method 233
setConnectTimeout method 233, 238
setDoInput method 233, 234, 238
setDoOutput method 233, 234, 238, 241, 243
setIfModifiedSince method 233, 235, 238
setReadTimeout method 233, 238
setRequestProperty method 233, 235, 238
setUseCaches method 233
URLDecoder class 247
decode method 247
URLEncoder class 247
encode method 247
URLs 231
attaching parameters to 239
connections via 231
encoding 239, 240
for databases 272
for namespace identifiers 182
redirecting 244
relative vs. absolute 163
URNs 231
US Letter paper 713
useLocale method 345, 347
User coordinates 673
User input 508
User Interface. See Graphical User Interface
User objects 615
User-Agent request parameter 244
userNodeForPackage method 490
of Preferences 494
userRoot method 490
of Preferences 494
Users
authentication of 409
preferences of 110
UTC 328
UTF-16 47, 67, 74, 79
byte order in 74
in regular expressions 136
native code and 744
UTF-8 74, 79
byte order in 75, 368
for text files 366
modified 79, 368, 744

V

validate method
of Component 509
validateObject method 132
Validation 162
activating 168
Value 180
value method
of XPathEvaluationResult 182
value-of element (XSLT) 204
valueChanged method 634, 635
of TreeSelectionListener 639
valueForPathChanged method 641
of TreeModel 647
ValueLayout interface 753
withName method 753
valueOf method (date/time legacy classes) 337
VARCHAR 271, 313
varHandle method
of MemoryLayout 765
Variable 641
toString method 642
Variable handle 762
Variables
binding 389
fields of 642
initializing 406
scope of 390
Variadic function
foreign 764
Vendor name, of a reader 689
Verifiers 405
Version number, of a reader 689
Versioning 129
Views 498
visitFile method
of SimpleFileVisitor 100
visitFile, visitFileFailed methods
of FileVisitor 98
of SimpleFileVisitor 99
visitFileFailed method
of SimpleFileVisitor 100

Visual Basic
event handling in 469

W

W3C 154, 186
walk method
of Files 95, 96
walkFileTree method 98
of Files 100
warning method
of ErrorHandler 169
Warnings
SQLWarning 280
WBMP 688
WeakReference class 643
Web applications 314
Web crawlers 186
with SAX parser 187
with StAX parser 190
Web pages
dynamic 383
separating class loaders for 401
WebRowSet interface 300
Weekends 322
Weeks 354, 355
weightx field
of GridBagConstraints 554
WHERE 269
Whitespace
ignoring, while parsing 156, 157
in e-mail URIs 240
in regular expressions 137
leading/trailing 508
Wilde, Oscar 339
Window class 454, 489
pack method 452, 454
Window listeners 475
windowActivated method
of WindowListener 476
windowClosed method
of WindowListener 476
WindowClosing 538
windowClosing method
of WindowListener 476
windowDeactivated method
of WindowListener 476
windowDeiconified method
of WindowListener 476
WindowEvent class 469, 475, 488
get*Xxx* methods of 489
windowFixed method
of Gatherers 45
WindowFocusListener interface, methods of 489
windowIconified method
of WindowListener 476
WindowListener interface 476
windowActivated method 476
windowClosed method 476
windowClosing method 476
windowDeactivated method 476
windowDeiconified method 476
windowIconified method 476
windowOpened method 476
WindowListener interface, methods of 475, 489
windowOpened method
of WindowListener 476
Windows
activating telnet in 211
authentication in 409
character encodings in 366
classpath in 272
glob syntax in 97
line feed in 70, 367
look-and-feel of 618
paths in 64, 87
resources in 369
using Microsoft compiler 745
Windows operating system
Alt+F4 keyboard shortcut in 538
pop-up menus in 536
registry in 490, 491
Windows-1252 366, 367
Windows. See Dialogs
windowStateChanged method
of WindowStateListener 476
WindowStateListener interface 476
windowStateChanged method 476
WindowStateListener interface, windowStateChanged method
490
with keyword 325
with method
of LocalDate 326
withDayOfMonth method
of LocalDate 324
withDayOfYear method
of LocalDate 324
withLocale method 332, 353
of DateTimeFormatter 336, 356
withMonth method
of LocalDate 324
withName method
of ValueLayout 753
withTargetLayout method
of AddressLayout 765
with*Xxx* method
of LocalTime 327
of Period 325
of ZonedDateTime 330
withYear method
of LocalDate 324
withZoneSameInstant method
of ZonedDateTime 330
withZoneSameLocal method
of ZonedDateTime 330
Words, in regular expressions 137
Working directory 64
wrap method
of ByteBuffer 107, 108
WritableByteChannel 109
write method 109
WritableByteChannel interface 224
WritableRaster class 697, 702
setDataElements method 699, 702
setPixel method 702
setPixel, setPixels methods 697
setPixels method 702
write method

of CipherOutputStream 438
of Files 89, 90
of ImageIO 688, 695
of ImageWriter 691, 696
of OutputStream 58, 60
of WritableByteChannel 109
of Writer 62
writeAttribute method 196
of XMLStreamWriter 201
writeBoolean method 78
of DataOutput 80
writeByte method 78
of DataOutput 80
writeCData method
of XMLStreamWriter 201
writeChar method 78, 81
of DataOutput 80
writeCharacters method 196
of XMLStreamWriter 201
writeChars method 78
of DataOutput 80
writeComment method
of XMLStreamWriter 201
writeDouble method 78, 114, 126
of DataOutput 80
writeDTD method
of XMLStreamWriter 201
writeEmptyElement method 196
of XMLStreamWriter 201
writeEndDocument method
of XMLStreamWriter 201
writeEndElement method
of XMLStreamWriter 201
writeEnd*Xxx* methods (XMLStreamWriter) 196
writeExternal method 127, 128
writeFixedString method 81, 82
writeFloat method 78
of DataOutput 80
writeInsert method 691
of ImageWriter 696
writeInt method 78, 81, 114
of DataOutput 80
writeLong method 78
of DataOutput 80
writeObject method
of HashSet 126
of ObjectOutputStream 113, 117, 126
Writer class 57, 62
write method 62
writeReplace method 128
writeShort method 78
of DataOutput 80
writeStartDocument method
of XMLStreamWriter 200
writeStartElement method
of XMLStreamWriter 201
writeStart*Xxx* methods (XMLStreamWriter) 196
writeString method 89
of Files 90
writeUTF method 78
of DataOutput 80

X

X.509 format 425
XHTML 151, 186
XML 149, 210
annotated version of the standard 151
attributes in 151
case sensitivity of 151
end and empty tags in 151
in databases 314
namespaces in 182
vs. HTML 151
XML catalogs 164
XML documents
DTDs in 151, 163
format of 150
generating 192
from non-XML legacy data 205
HTML files from 201, 204
plain text from 204, 206
with StAX 196
locating information in 178
malformed 197
parsing 154
structure of 151, 155, 162
validating 162
with/without namespaces 192, 193
XML Schema 162, 163, 170
attributes in 172
documentation for 182
parsing with 173
referencing in XML documents 170
repeated elements in 172
type definitions in 171, 172
XMLInputFactory class 191
createXMLStreamReader method 191
newInstance method 191
setProperty method 190, 191
Xmlns 183
XMLOutputFactory class 200
createXMLStreamWriter method 196, 200
newInstance method 196, 200
XMLReader interface 210
implementing 205
parse method 210
setContentHandler method 210
XMLStreamReader interface 192
getAttributeCount method 192
getAttributeLocalName method 192
getAttributeName method 192
getAttributeValue method 192
getAttribute*Xxx* methods 190
getLocalName method 192
getName method 192
getText method 192
hasNext method 192
isCharacters method 192
isEndElement method 192
isStartElement method 192
isWhiteSpace method 192
next method 192
XMLStreamWriter interface 196, 200
close method 201
not autocloseable 197

setDefaultNamespace method 200
setPrefix method 200
writeAttribute method 196, 201
writeCData method 201
writeCharacters method 196, 201
writeComment method 201
writeDTD method 201
writeEmptyElement method 196, 201
writeEndDocument method 201
writeEndElement method 201
writeEnd*Xxx* methods 196
writeStartDocument method 200
writeStartElement method 201
writeStart*Xxx* methods 196
XOR composition rule 682
XPath interface 178, 181
count function 179
evaluate method 179, 180, 181
evaluateExpression method 179, 180, 181
XPathEvaluationResult interface 182
type method 182
value method 180, 182
XPathFactory class 181
newInstance method 179, 181
newXpath method 181
XPathNodes interface 179
xs:, xsd: prefixes (XML Schema) 171
Xsd:attribute 172
Xsd:choice 172
Xsd:complexType 171
Xsd:element 171
Xsd:enumeration 171
Xsd:schema 173
Xsd:sequence 172
Xsd:simpleType 171
Xsl:apply-templates 203
Xsl:output 203
Xsl:template 203
Xsl:value-of 204
XSLT 193, 201
copying attribute values in 203
templates in 203
XSLT processors 201

Y

Year, YearMonth classes 323

Z

Z (boolean), type code 119
ZIP archives 84
reading 66, 84
writing 84
Zip code lookup 242
ZIP file systems 100
ZipEntry class 86
Constructor 86
getCrc method 86
getName method 86
getSize method 86
isDirectory method 86
setCrc method 86
setMethod method 86
setSize method 86
ZipInputStream class 61, 84, 85
closeEntry method 84, 85
Constructor 85
getNextEntry method 84, 85
read method 84
ZipOutputStream class 61, 84, 85
closeEntry method 84, 85
Constructor 85
putNextEntry method 84, 85
setLevel method 85
setMethod method 86
ZonedDateTime class 327, 330, 336, 356
format method 353
from method 336
getDayOfMonth method 330
getDayOfWeek method 330
getDayOfYear method 330
getHour method 331
getMinute method 331
getMonth method 331
getMonthValue method 331
getNano method 331
getOffset method 331
getSecond method 331
getYear method 331
isAfter method 331
isBefore method 331
legacy classes and 336
minus method 330
minus*Xxx* method 330
now method 330
of method 330
of, ofInstant methods 328
ofInstant method 330
parse method 336, 354, 356
plus method 330
plus*Xxx* method 330
toInstant method 328, 331
toLocalDate method 331
toLocalDateTime method 331
toLocalTime method 331
with*Xxx* method 330
withZoneSameInstant method 330
withZoneSameLocal method 330
ZoneId class 337
getAvailableZoneIds method 328
of method 328