The Core Series is designed to provide you – the experienced programmer – with the essential information you need to quickly learn and apply the latest, most important technologies.

Authors in The Core Series are seasoned professionals who have pioneered the use of these technologies to achieve tangible results in real-world settings. These experts:
- Share their practical experiences
- Support their instruction with real-world examples
- Provide an accelerated, highly effective path to learning the subject at hand

The resulting book is a no-nonsense tutorial and thorough reference that allows you to quickly produce robust, production-quality code.

Visit informit.com/coreseries for a complete list of available publications.
Preface ........................................................................................................... xix
Acknowledgments ........................................................................................ xxv

Chapter 1: An Introduction to Java .............................................................. 1
  1.1 Java As a Programming Platform .......................................................... 1
  1.2 The Java “White Paper” Buzzwords .................................................... 2
    1.2.1 Simple .................................................................................. 3
    1.2.2 Object-Oriented ................................................................. 4
    1.2.3 Network-Savvy ................................................................... 4
    1.2.4 Robust ............................................................................... 5
    1.2.5 Secure .............................................................................. 5
    1.2.6 Architecture-Neutral .......................................................... 6
    1.2.7 Portable ............................................................................ 7
    1.2.8 Interpreted ....................................................................... 7
    1.2.9 High-Performance ............................................................. 8
    1.2.10 Multithreaded ................................................................. 8
    1.2.11 Dynamic ........................................................................... 8
  1.3 Java Applets and the Internet .............................................................. 9
  1.4 A Short History of Java ...................................................................... 10
  1.5 Common Misconceptions about Java ............................................... 13

Chapter 2: The Java Programming Environment ...................................... 17
  2.1 Installing the Java Development Kit .................................................. 18
    2.1.1 Downloading the JDK ......................................................... 18
    2.1.2 Setting the Executable Path ............................................... 20
    2.1.3 Installing the Library Source and Documentation ............... 22
    2.1.4 Installing the Core Java Program Examples ....................... 23
    2.1.5 Navigating the Java Directories ........................................... 24
  2.2 Choosing a Development Environment ............................................ 24
  2.3 Using the Command-Line Tools ....................................................... 25
# Chapter 3: Fundamental Programming Structures in Java

3.1 A Simple Java Program ............................................................. 42
3.2 Comments ............................................................................... 45
3.3 Data Types ................................................................................. 46
    3.3.1 Integer Types .............................................................. 47
    3.3.2 Floating-Point Types .................................................. 48
    3.3.3 The char Type ............................................................ 49
    3.3.4 The boolean Type ....................................................... 51
3.4 Variables ....................................................................................... 52
    3.4.1 Initializing Variables .................................................... 53
    3.4.2 Constants .................................................................. 54
3.5 Operators ....................................................................................... 55
    3.5.1 Increment and Decrement Operators ............................... 56
    3.5.2 Relational and boolean Operators ................................ 57
    3.5.3 Bitwise Operators ....................................................... 58
    3.5.4 Mathematical Functions and Constants ......................... 59
    3.5.5 Conversions between Numeric Types ................................ 60
    3.5.6 Casts ........................................................................ 61
    3.5.7 Parentheses and Operator Hierarchy ............................... 62
    3.5.8 Enumerated Types ....................................................... 63
3.6 Strings ......................................................................................... 64
    3.6.1 Substrings ................................................................ 64
    3.6.2 Concatenation ............................................................ 64
    3.6.3 Strings Are Immutable ................................................ 65
    3.6.4 Testing Strings for Equality ......................................... 67
    3.6.5 Empty and Null Strings ............................................. 68
    3.6.6 Code Points and Code Units ........................................ 68
    3.6.7 The String API ........................................................... 69
    3.6.8 Reading the Online API Documentation ......................... 72
    3.6.9 Building Strings .......................................................... 74
Chapter 5: Inheritance ................................................................. 203
5.1 Classes, Superclasses, and Subclasses ........................................ 204
  5.1.1 Inheritance Hierarchies ................................................. 212
  5.1.2 Polymorphism .............................................................. 213
  5.1.3 Dynamic Binding .......................................................... 214
  5.1.4 Preventing Inheritance: Final Classes and Methods .......... 217
  5.1.5 Casting .................................................................... 218
  5.1.6 Abstract Classes ......................................................... 221
  5.1.7 Protected Access ........................................................ 227
5.2 Object: The Cosmic Superclass ................................................. 228
  5.2.1 The equals Method ....................................................... 228
  5.2.2 Equality Testing and Inheritance ................................... 230
  5.2.3 The hashCode Method ................................................. 234
  5.2.4 The toString Method .................................................. 236
5.3 Generic Array Lists ................................................................ 243
  5.3.1 Accessing Array List Elements ....................................... 246
  5.3.2 Compatibility between Typed and Raw Array Lists .......... 249
5.4 Object Wrappers and Autoboxing ............................................ 251
5.5 Methods with a Variable Number of Parameters ..................... 254
5.6 Enumeration Classes .......................................................... 256
5.7 Reflection ............................................................................. 258
  5.7.1 The Class Class .......................................................... 259
  5.7.2 A Primer on Catching Exceptions ................................. 261
  5.7.3 Using Reflection to Analyze the Capabilities of Classes .... 263
  5.7.4 Using Reflection to Analyze Objects at Runtime .......... 269
  5.7.5 Using Reflection to Write Generic Array Code ............. 274
  5.7.6 Invoking Arbitrary Methods ........................................ 278
5.8 Design Hints for Inheritance .................................................. 282

Chapter 6: Interfaces and Inner Classes ........................................ 285
6.1 Interfaces ............................................................................. 286
  6.1.1 Properties of Interfaces ............................................... 292
  6.1.2 Interfaces and Abstract Classes ..................................... 294
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 9: User Interface Components with Swing</td>
<td>425</td>
</tr>
<tr>
<td>9.1 Swing and the Model-View-Controller Design Pattern</td>
<td>426</td>
</tr>
<tr>
<td>9.1.1 Design Patterns</td>
<td>426</td>
</tr>
<tr>
<td>9.1.2 The Model-View-Controller Pattern</td>
<td>428</td>
</tr>
<tr>
<td>9.1.3 A Model-View-Controller Analysis of Swing Buttons</td>
<td>432</td>
</tr>
<tr>
<td>9.2 Introduction to Layout Management</td>
<td>433</td>
</tr>
<tr>
<td>9.2.1 Border Layout</td>
<td>437</td>
</tr>
<tr>
<td>9.2.2 Grid Layout</td>
<td>439</td>
</tr>
<tr>
<td>9.3 Text Input</td>
<td>443</td>
</tr>
<tr>
<td>9.3.1 Text Fields</td>
<td>444</td>
</tr>
<tr>
<td>9.3.2 Labels and Labeling Components</td>
<td>446</td>
</tr>
<tr>
<td>9.3.3 Password Fields</td>
<td>447</td>
</tr>
<tr>
<td>9.3.4 Text Areas</td>
<td>448</td>
</tr>
<tr>
<td>9.3.5 Scroll Panes</td>
<td>449</td>
</tr>
<tr>
<td>9.4 Choice Components</td>
<td>452</td>
</tr>
<tr>
<td>9.4.1 Checkboxes</td>
<td>452</td>
</tr>
<tr>
<td>9.4.2 Radio Buttons</td>
<td>454</td>
</tr>
<tr>
<td>9.4.3 Borders</td>
<td>458</td>
</tr>
<tr>
<td>9.4.4 Combo Boxes</td>
<td>462</td>
</tr>
<tr>
<td>9.4.5 Sliders</td>
<td>466</td>
</tr>
<tr>
<td>9.5 Menus</td>
<td>473</td>
</tr>
<tr>
<td>9.5.1 Menu Building</td>
<td>473</td>
</tr>
<tr>
<td>9.5.2 Icons in Menu Items</td>
<td>476</td>
</tr>
<tr>
<td>9.5.3 Checkbox and Radio Button Menu Items</td>
<td>477</td>
</tr>
<tr>
<td>9.5.4 Pop-Up Menus</td>
<td>479</td>
</tr>
<tr>
<td>9.5.5 Keyboard Mnemonics and Accelerators</td>
<td>480</td>
</tr>
<tr>
<td>9.5.6 Enabling and Disabling Menu Items</td>
<td>483</td>
</tr>
<tr>
<td>9.5.7 Toolbars</td>
<td>488</td>
</tr>
<tr>
<td>9.5.8 Tooltips</td>
<td>490</td>
</tr>
<tr>
<td>9.6 Sophisticated Layout Management</td>
<td>492</td>
</tr>
<tr>
<td>9.6.1 The Grid Bag Layout</td>
<td>494</td>
</tr>
<tr>
<td>9.6.1.1 The gridx, gridy, gridwidth, and gridheight Parameters</td>
<td>496</td>
</tr>
<tr>
<td>9.6.1.2 Weight Fields</td>
<td>496</td>
</tr>
<tr>
<td>9.6.1.3 The fill and anchor Parameters</td>
<td>497</td>
</tr>
<tr>
<td>9.6.1.4 Padding</td>
<td>497</td>
</tr>
</tbody>
</table>
9.6.1.5 Alternative Method to Specify the gridx, gridy, gridwidth, and gridheight Parameters ............................................ 497
9.6.1.6 A Helper Class to Tame the Grid Bag Constraints ......................................................................................... 499
9.6.2 Group Layout .................................................................................................................................................. 505
9.6.3 Using No Layout Manager ......................................................................................................................... 516
9.6.4 Custom Layout Managers ......................................................................................................................... 516
9.6.5 Traversal Order ........................................................................................................................................ 521
9.7 Dialog Boxes ................................................................................................................................................. 522
  9.7.1 Option Dialogs ....................................................................................................................................... 523
  9.7.2 Creating Dialogs .................................................................................................................................... 533
  9.7.3 Data Exchange ...................................................................................................................................... 538
  9.7.4 File Dialogs ........................................................................................................................................... 545
  9.7.5 Color Choosers ...................................................................................................................................... 557

Chapter 10: Deploying Applications and Applets ................................................................. 565
  10.1 JAR Files .................................................................................................................................................. 566
      10.1.1 The Manifest .................................................................................................................................. 567
      10.1.2 Executable JAR Files .................................................................................................................. 568
      10.1.3 Resources ...................................................................................................................................... 569
      10.1.4 Sealing ........................................................................................................................................... 573
  10.2 Java Web Start ......................................................................................................................................... 574
      10.2.1 The Sandbox .................................................................................................................................. 578
      10.2.2 Signed Code .................................................................................................................................. 579
      10.2.3 The JNLP API .............................................................................................................................. 582
  10.3 Applets ...................................................................................................................................................... 591
      10.3.1 A Simple Applet ........................................................................................................................... 591
          10.3.1.1 Converting Applications to Applets ..................................................................................... 595
      10.3.2 The applet HTML Tag and Its Attributes .................................................................................... 596
      10.3.3 The object Tag .............................................................................................................................. 599
      10.3.4 Use of Parameters to Pass Information to Applets .................................................................... 600
      10.3.5 Accessing Image and Audio Files ................................................................................................. 606
      10.3.6 The Applet Context ....................................................................................................................... 607
          10.3.6.1 Inter-Applet Communication ............................................................................................ 608
          10.3.6.2 Displaying Items in the Browser ......................................................................................... 609
10.4 Storage of Application Preferences .......................................................... 610
10.4.1 Property Maps ..................................................................................... 611
10.4.2 The Preferences API ......................................................................... 616

Chapter 11: Exceptions, Assertions, Logging, and Debugging .............. 625
11.1 Dealing with Errors .............................................................................. 626
  11.1.1 The Classification of Exceptions ....................................................... 628
  11.1.2 Declaring Checked Exceptions ....................................................... 630
  11.1.3 How to Throw an Exception ............................................................... 632
  11.1.4 Creating Exception Classes ............................................................... 634
11.2 Catching Exceptions ........................................................................... 635
  11.2.1 Catching Multiple Exceptions ........................................................... 637
  11.2.2 Rethrowing and Chaining Exceptions ............................................ 639
  11.2.3 The finally Clause ........................................................................... 640
  11.2.4 The Try-with-Resources Statement ............................................... 644
  11.2.5 Analyzing Stack Trace Elements ..................................................... 646
11.3 Tips for Using Exceptions .................................................................... 649
11.4 Using Assertions .................................................................................. 653
  11.4.1 Assertion Enabling and Disabling ..................................................... 654
  11.4.2 Using Assertions for Parameter Checking ...................................... 655
  11.4.3 Using Assertions for Documenting Assumptions ............................ 656
11.5 Logging ............................................................................................... 657
  11.5.1 Basic Logging .................................................................................. 658
  11.5.2 Advanced Logging .......................................................................... 658
  11.5.3 Changing the Log Manager Configuration .................................... 661
  11.5.4 Localization .................................................................................... 662
  11.5.5 Handlers ......................................................................................... 663
  11.5.6 Filters ............................................................................................ 667
  11.5.7 Formatters ..................................................................................... 667
  11.5.8 A Logging Recipe ........................................................................... 668
11.6 Debugging Tips .................................................................................... 677
11.7 Tips for Troubleshooting GUI Programs ............................................ 682
  11.7.1 Letting the AWT Robot Do the Work .............................................. 686
11.8 Using a Debugger ................................................................................ 690
Chapter 12: Generic Programming ............................................................... 697
  12.1 Why Generic Programming? ............................................................... 698
    12.1.1 Who Wants to Be a Generic Programmer? ............................ 699
  12.2 Defining a Simple Generic Class .................................................... 700
  12.3 Generic Methods ............................................................................. 702
  12.4 Bounds for Type Variables ............................................................ 704
  12.5 Generic Code and the Virtual Machine ........................................... 706
    12.5.1 Translating Generic Expressions ........................................... 708
    12.5.2 Translating Generic Methods .............................................. 708
    12.5.3 Calling Legacy Code ............................................................. 711
  12.6 Restrictions and Limitations ............................................................ 712
    12.6.1 Type Parameters Cannot Be Instantiated with Primitive Types ........................................... 712
    12.6.2 Runtime Type Inquiry Only Works with Raw Types .......... 712
    12.6.3 You Cannot Create Arrays of Parameterized Types .......... 713
    12.6.4 Varargs Warnings ................................................................. 713
    12.6.5 You Cannot Instantiate Type Variables .................................. 715
    12.6.6 Type Variables Are Not Valid in Static Contexts of Generic Classes ........................................... 717
    12.6.7 You Cannot Throw or Catch Instances of a Generic Class ... 717
    12.6.7.1 You Can Defeat Checked Exception Checking ...... 718
    12.6.8 Beware of Clashes after Erasure .............................................. 720
  12.7 Inheritance Rules for Generic Types .................................................... 721
  12.8 Wildcard Types ................................................................................. 723
    12.8.1 Supertype Bounds for Wildcards ........................................... 725
    12.8.2 Unbounded Wildcards ............................................................. 728
    12.8.3 Wildcard Capture ................................................................. 728
  12.9 Reflection and Generics ..................................................................... 731
    12.9.1 Using Class<T> Parameters for Type Matching ......................... 732
    12.9.2 Generic Type Information in the Virtual Machine ..................... 733

Chapter 13: Collections ............................................................................. 741
  13.1 Collection Interfaces .......................................................................... 741
    13.1.1 Separating Collection Interfaces and Implementation ............ 742
    13.1.2 Collection and Iterator Interfaces in the Java Library .......... 745
      13.1.2.1 Iterators ............................................................................. 745
13.1.2.2 Removing Elements ................................................... 748
13.1.2.3 Generic Utility Methods ............................................ 748

13.2 Concrete Collections ................................................... 751
  13.2.1 Linked Lists ................................................................. 752
  13.2.2 Array Lists ................................................................. 762
  13.2.3 Hash Sets ................................................................. 763
  13.2.4 Tree Sets ................................................................. 767
  13.2.5 Object Comparison .................................................... 768
  13.2.6 Queues and Deques ................................................... 774
  13.2.7 Priority Queues ......................................................... 776
  13.2.8 Maps ........................................................................ 777
  13.2.9 Specialized Set and Map Classes ............................... 782
    13.2.9.1 Weak Hash Maps .................................................. 782
    13.2.9.2 Linked Hash Sets and Maps .................................. 783
    13.2.9.3 Enumeration Sets and Maps ................................. 785
    13.2.9.4 Identity Hash Maps .............................................. 785

13.3 The Collections Framework ........................................... 787
  13.3.1 Views and Wrappers .................................................... 792
    13.3.1.1 Lightweight Collection Wrappers .......................... 793
    13.3.1.2 Subranges .......................................................... 794
    13.3.1.3 Unmodifiable Views ............................................. 794
    13.3.1.4 Synchronized Views ............................................ 796
    13.3.1.5 Checked Views .................................................. 796
    13.3.1.6 A Note on Optional Operations ............................. 797
  13.3.2 Bulk Operations ........................................................ 799
  13.3.3 Converting between Collections and Arrays .................. 800

13.4 Algorithms ................................................................. 801
  13.4.1 Sorting and Shuffling .................................................. 802
  13.4.2 Binary Search ............................................................. 805
  13.4.3 Simple Algorithms ..................................................... 806
  13.4.4 Writing Your Own Algorithms ..................................... 808

13.5 Legacy Collections ........................................................ 810
  13.5.1 The Hashable Class ..................................................... 810
  13.5.2 Enumerations ............................................................ 810
  13.5.3 Property Maps .......................................................... 811
Chapter 14: Multithreading ................................................................. 819

14.1 What Are Threads? ................................................................. 820
  14.1.1 Using Threads to Give Other Tasks a Chance .................. 827
14.2 Interrupting Threads ............................................................... 833
14.3 Thread States .......................................................................... 836
  14.3.1 New Threads ................................................................. 836
  14.3.2 Runnable Threads .......................................................... 836
  14.3.3 Blocked and Waiting Threads ......................................... 837
  14.3.4 Terminated Threads ...................................................... 839
14.4 Thread Properties ................................................................. 839
  14.4.1 Thread Priorities ............................................................ 840
  14.4.2 Daemon Threads .......................................................... 841
  14.4.3 Handlers for Uncaught Exceptions ................................. 841
14.5 Synchronization ..................................................................... 843
  14.5.1 An Example of a Race Condition ................................. 843
  14.5.2 The Race Condition Explained ...................................... 848
  14.5.3 Lock Objects .............................................................. 850
  14.5.4 Condition Objects ....................................................... 854
  14.5.5 The synchronized Keyword ......................................... 859
  14.5.6 Synchronized Blocks .................................................. 864
  14.5.7 The Monitor Concept .................................................. 865
  14.5.8 Volatile Fields ............................................................ 866
  14.5.9 Final Variables ........................................................... 867
  14.5.10 Atomics ................................................................. 868
  14.5.11 Deadlocks ............................................................... 868
  14.5.12 Thread-Local Variables ............................................. 871
  14.5.13 Lock Testing and Timeouts ........................................ 873
  14.5.14 Read/Write Locks ...................................................... 874
  14.5.15 Why the stop and suspend Methods Are Deprecated .... 875
14.6 Blocking Queues ................................................................... 877
14.7 Thread-Safe Collections .......................................................... 886
  14.7.1 Efficient Maps, Sets, and Queues ................................. 886
To the Reader

In late 1995, the Java programming language burst onto the Internet scene and gained instant celebrity status. The promise of Java technology was that it would become the universal glue that connects users with information wherever it comes from—web servers, databases, information providers, or any other imaginable source. Indeed, Java is in a unique position to fulfill this promise. It is an extremely solidly engineered language that has gained acceptance by all major vendors, except for Microsoft. Its built-in security and safety features are reassuring both to programmers and to the users of Java programs. Java even has built-in support for advanced programming tasks, such as network programming, database connectivity, and multithreading.

Since 1995, eight major revisions of the Java Development Kit have been released. Over the course of the last 17 years, the Application Programming Interface (API) has grown from about 200 to over 3,000 classes. The API now spans such diverse areas as user interface construction, database management, internationalization, security, and XML processing.

The book you have in your hands is the first volume of the ninth edition of Core Java™. Each edition closely followed a release of the Java Development Kit, and each time, we rewrote the book to take advantage of the newest Java features. This edition has been updated to reflect the features of Java Standard Edition (SE) 7.

As with the previous editions of this book, we still target serious programmers who want to put Java to work on real projects. We think of you, our reader, as a programmer with a solid background in a programming language other than Java, and we assume that you don’t like books filled with toy examples (such as toasters, zoo animals, or “nervous text”). You won’t find any of these in our book. Our goal is to enable you to fully understand the Java language and library, not to give you an illusion of understanding.

In this book you will find lots of sample code demonstrating almost every language and library feature that we discuss. We keep the sample programs purposefully simple to focus on the major points, but, for the most part, they aren’t fake and they don’t cut corners. They should make good starting points for your own code.
We assume you are willing, even eager, to learn about all the advanced features that Java puts at your disposal. For example, we give you a detailed treatment of

- Object-oriented programming
- Reflection and proxies
- Interfaces and inner classes
- The event listener model
- Graphical user interface design with the Swing UI toolkit
- Exception handling
- Generic programming
- The collections framework
- Concurrency

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 no longer possible. Hence, we decided to break up the book into two volumes. The first volume, which you hold in your hands, concentrates on the fundamental concepts of the Java language, along with the basics of user-interface programming. The second volume, Core Java™, Volume II—Advanced Features (forthcoming, ISBN: 978-0-13-708160-8), goes further into the enterprise features and advanced user-interface programming. It includes detailed discussions of

- Files and streams
- Distributed objects
- Databases
- Advanced GUI components
- Native methods
- XML processing
- Network programming
- Advanced graphics
- Internationalization
- JavaBeans
- Annotations

When writing a book, errors and inaccuracies are inevitable. We’d very much like to know about them. But, of course, we’d prefer to learn about each of them only once. We have put up a list of frequently asked questions, bugs fixes, and workarounds on a web page at http://horstmann.com/corejava. Strategically placed at the end of the errata page (to encourage you to read through it first) is a form you can use to report bugs and suggest improvements. Please don’t be
disappointed if we don’t answer every query or don’t get back to you immediately. We do read all e-mail and appreciate your input to make future editions of this book clearer and more informative.

A Tour of This Book

Chapter 1 gives an overview of the capabilities of Java that set it apart from other programming languages. We explain what the designers of the language set out to do and to what extent they succeeded. Then, we give a short history of how Java came into being and how it has evolved.

In Chapter 2, we tell you how to download and install the JDK and the program examples for this book. Then we guide you through compiling and running three typical Java programs—a console application, a graphical application, and an applet—using the plain JDK, a Java-enabled text editor, and a Java IDE.

Chapter 3 starts the discussion of the Java language. In this chapter, we cover the basics: variables, loops, and simple functions. If you are a C or C++ programmer, this is smooth sailing because the syntax for these language features is essentially the same as in C. If you come from a non-C background such as Visual Basic, you will want to read this chapter carefully.

Object-oriented programming (OOP) is now in the mainstream of programming practice, and Java is an object-oriented programming language. Chapter 4 introduces encapsulation, the first of two fundamental building blocks of object orientation, and the Java language mechanism to implement it—that is, classes and methods. In addition to the rules of the Java language, we also give advice on sound OOP design. Finally, we cover the marvelous javadoc tool that formats your code comments as a set of hyperlinked web pages. If you are familiar with C++, you can browse through this chapter quickly. Programmers coming from a non-object-oriented background should expect to spend some time mastering the OOP concepts before going further with Java.

Classes and encapsulation are only one part of the OOP story, and Chapter 5 introduces the other—namely, inheritance. Inheritance lets you take an existing class and modify it according to your needs. This is a fundamental technique for programming in Java. The inheritance mechanism in Java is quite similar to that in C++. Once again, C++ programmers can focus on the differences between the languages.

Chapter 6 shows you how to use Java’s notion of an interface. Interfaces let you go beyond the simple inheritance model of Chapter 5. Mastering interfaces allows you to have full access to the power of Java’s completely object-oriented approach
to programming. We also cover a useful technical feature of Java called *inner classes*. Inner classes help make your code cleaner and more concise.

In **Chapter 7**, we begin application programming in earnest. Every Java programmer should know a bit about GUI programming, and this volume contains the basics. We 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.

**Chapter 8** is a detailed discussion of the event model of the AWT, the *abstract window toolkit*. You’ll see how to write code that responds to events, such as mouse clicks or key presses. Along the way you’ll see how to handle basic GUI elements like buttons and panels.

**Chapter 9** 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. However, some of the more advanced components are discussed in Volume II.

**Chapter 10** shows you how to deploy your programs, either as applications or applets. We describe how to package programs in JAR files, and how to deliver applications over the Internet with the Java Web Start and applet mechanisms. Finally, we explain how Java programs can store and retrieve configuration information once they have been deployed.

**Chapter 11** discusses *exception handling*—Java’s robust mechanism to deal with the fact that bad things can happen to good programs. Exceptions give you an efficient way of separating the normal processing code from the error handling. Of course, even after hardening your program by handling all exceptional conditions, it still might fail to work as expected. In the second half of this chapter, we give you a large number of useful debugging tips. Finally, we guide you through a sample debugging session.

**Chapter 12** gives an overview of generic programming—a major advance of Java SE 5.0. Generic programming makes your programs easier to read and safer. We show you how to use strong typing and remove unsightly and unsafe casts, and how to deal with the complexities that arise from the need to stay compatible with older versions of Java.

The topic of **Chapter 13** is the collections framework of the Java platform. Whenever you want to collect multiple objects and retrieve them later, you should use a collection that is best suited for your circumstances, instead of just tossing the elements into an array. This chapter shows you how to take advantage of the standard collections that are prebuilt for your use.
Chapter 14 finishes the book with a discussion of multithreading, which enables you to program tasks to be done in parallel. (A thread is a flow of control within a program.) We show you how to set up threads and how to deal with thread synchronization. Multithreading has changed a great deal in Java SE 5.0, and we tell you all about the new mechanisms.

The Appendix lists the reserved words of the Java language.

Conventions

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

### NOTE:
Notes are tagged with “note” icons that look like this.

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

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

### C++ NOTE:
There are many C++ notes that explain the differences between Java and C++. You can skip over them if you don’t have a background in C++ or if you consider your experience with that language a bad dream of which you’d rather not be reminded.

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

**Application Programming Interface 1.2**
Programs whose source code is on the book’s companion web site are listed as listings, for instance

```
Listing 1.1  InputTest/InputTest.java
```

**Sample Code**

The web site for this book at http://horstmann.com/corejava contains all sample code from the book, in compressed form. You can expand the file either with one of the familiar unzipping programs or simply with the jar utility that is part of the Java Development Kit. See Chapter 2 for more information about installing the Java Development Kit and the sample code.
Writing a book is always a monumental effort, and rewriting it doesn’t seem to be much easier, especially with the continuous change in Java technology. Making a book a reality takes many dedicated people, and it is my great pleasure to acknowledge the contributions of the entire Core Java team.

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

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), Alec Beaton (IBM), Cliff Berg, Joshua Bloch, David Brown, Corky Cartwright, Frank Cohen (PushToTest), Chris Crane (devXsolution), Dr. Nicholas J. De Lillo (Manhattan College), Rakesh Dhoopar (Oracle), 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), Marty Hall (coreservlets.com, Inc.), Vincent Hardy (Adobe Systems), Dan Harkey (San Jose State University), William Higgins (IBM), Vladimir Ivanovic (PointBase), Jerry Jackson (CA Technologies), Tim Kimmet (Walmart), Chris Laffra, Charlie Lai (Apple), Angelika Langer, Doug Langston, Hang Lau (McGill University), Mark Lawrence, Doug Lea (SUNY Oswego), Gregory Longshore, Bob Lynch (Lynch Associates), Philip Milne (consultant), Mark Morrissey (The Oregon Graduate Institute), Mahesh Neelakanta (Florida Atlantic University), Hao Pham, Paul Philion, Blake Ragsdell, Stuart Reges (University of Arizona), Rich Rosen (Interactive Data Corporation), Peter Sanders (ESSI University, Nice, France), Dr. Paul Sanghera (San Jose State University and University of Arizona), Rich Rosen (Interactive Data Corporation), Peter Sanders (ESSI University, Nice, France), Dr. Paul Sanghera (San Jose State University and
Brooks College), Paul Sevinc (Teamup AG), Devang Shah (Sun Microsystems), Bradley A. Smith, Steven Stelting (Oracle), Christopher Taylor, Luke Taylor (Valtech), George Thiruvathukal, Kim Topley (StreamingEdge), Janet Traub, Paul Tyma (consultant), Peter van der Linden (Motorola Mobile Devices), Burt Walsh, Dan Xu (Oracle), and John Zavgren (Oracle).

_Cay Horstmann_

_San Francisco, California_

_Scptember 2012_
You have now seen all the basic tools for object-oriented programming in Java. This chapter shows you several advanced techniques that are commonly used. Despite their less obvious nature, you will need to master them to complete your Java tool chest.

The first technique, called interfaces, is a way of describing what classes should do, without specifying how they should do it. A class can implement one or more interfaces. You can then use objects of these implementing classes whenever conformance to the interface is required. After we cover interfaces, we take up cloning an object (or deep copying, as it is sometimes called). A clone of an object is a new object that has the same state as the original. In particular, you can modify the clone without affecting the original.

Next, we move on to the mechanism of inner classes. Inner classes are technically somewhat complex—they are defined inside other classes, and their methods can access the fields of the surrounding class. Inner classes are useful when you design collections of cooperating classes. In particular, inner classes enable you to write concise, professional looking code to handle GUI events.
This chapter concludes with a discussion of proxies, objects that implement arbitrary interfaces. A proxy is a very specialized construct that is useful for building system-level tools. You can safely skip that section on first reading.

### 6.1 Interfaces

In the Java programming language, an interface is not a class but a set of requirements for the classes that want to conform to the interface.

Typically, the supplier of some service states: “If your class conforms to a particular interface, then I’ll perform the service.” Let’s look at a concrete example. The sort method of the `Arrays` class promises to sort an array of objects, but under one condition: The objects must belong to classes that implement the `Comparable` interface.

Here is what the `Comparable` interface looks like:

```java
public interface Comparable {
    int compareTo(Object other);
}
```

This means that any class that implements the `Comparable` interface is required to have a `compareTo` method, and the method must take an `Object` parameter and return an integer.

**NOTE:** As of Java SE 5.0, the `Comparable` interface has been enhanced to be a generic type.

```java
public interface Comparable<T> {
    int compareTo(T other); // parameter has type T
}
```

For example, a class that implements `Comparable<Employee>` must supply a method

```java
int compareTo(Employee other)
```

You can still use the “raw” `Comparable` type without a type parameter, but then you have to manually cast the parameter of the `compareTo` method to the desired type.

All methods of an interface are automatically `public`. For that reason, it is not necessary to supply the keyword `public` when declaring a method in an interface.

Of course, there is an additional requirement that the interface cannot spell out: When calling `x.compareTo(y)`, the `compareTo` method must actually be able to compare the two objects and return an indication whether `x` or `y` is larger. The method is
supposed to return a negative number if \( x \) is smaller than \( y \), zero if they are equal, and a positive number otherwise.

This particular interface has a single method. Some interfaces have multiple methods. As you will see later, interfaces can also define constants. What is more important, however, is what interfaces cannot supply. Interfaces never have instance fields, and the methods are never implemented in the interface. Supplying instance fields and method implementations is the job of the classes that implement the interface. You can think of an interface as being similar to an abstract class with no instance fields. However, there are some differences between these two concepts—we look at them later in some detail.

Now suppose we want to use the sort method of the Arrays class to sort an array of Employee objects. Then the Employee class must implement the Comparable interface.

To make a class implement an interface, you carry out two steps:

1. You declare that your class intends to implement the given interface.
2. You supply definitions for all methods in the interface.

To declare that a class implements an interface, use the implements keyword:

```java
class Employee implements Comparable
```

Of course, now the Employee class needs to supply the compareTo method. Let’s suppose that we want to compare employees by their salary. Here is an implementation of the compareTo method:

```java
public int compareTo(Object otherObject)
{
    Employee other = (Employee) otherObject;
    return Double.compare(salary, other.salary);
}
```

Here, we use the static Double.compare method that returns a negative if the first argument is less than the second argument, 0 if they are equal, and a positive value otherwise.

---

**CAUTION:** In the interface declaration, the compareTo method was not declared public because all methods in an interface are automatically public. However, when implementing the interface, you must declare the method as public. Otherwise, the compiler assumes that the method has package visibility—the default for a class. The compiler then complains that you’re trying to supply a weaker access privilege.
As of Java SE 5.0, we can do a little better. We’ll implement the `Comparable<Employee>` interface type instead.

```java
class Employee implements Comparable<Employee> {
    public int compareTo(Employee other) {
        return Double.compare(salary, other.salary);
    }
    ...
}
```

Note that the unsightly cast of the `Object` parameter has gone away.

**TIP:** The `compareTo` method of the `Comparable` interface returns an integer. If the objects are not equal, it does not matter what negative or positive value you return. This flexibility can be useful when you are comparing integer fields. For example, suppose each employee has a unique integer `id` and you want to sort by the employee ID number. Then you can simply return `id - other.id`. That value will be some negative value if the first ID number is less than the other, 0 if they are the same ID, and some positive value otherwise. However, there is one caveat: The range of the integers must be small enough so that the subtraction does not overflow. If you know that the IDs are not negative or that their absolute value is at most `(Integer.MAX_VALUE - 1) / 2`, you are safe.

Of course, the subtraction trick doesn’t work for floating-point numbers. The difference `salary - other.salary` can round to 0 if the salaries are close together but not identical. The call `Double.compare(x, y)` simply returns `-1` if `x < y` or `1` if `x > y`.

Now you saw what a class must do to avail itself of the sorting service—it must implement a `compareTo` method. That’s eminently reasonable. There needs to be some way for the `sort` method to compare objects. But why can’t the `Employee` class simply provide a `compareTo` method without implementing the `Comparable` interface?

The reason for interfaces is that the Java programming language is *strongly typed*. When making a method call, the compiler needs to be able to check that the method actually exists. Somewhere in the `sort` method will be statements like this:

```java
if (a[i].compareTo(a[j]) > 0) {
    // rearrange a[i] and a[j]
    ...
}
```

The compiler must know that `a[i]` actually has a `compareTo` method. If `a` is an array of `Comparable` objects, then the existence of the method is assured because every class that implements the `Comparable` interface must supply the method.
NOTE: You would expect that the sort method in the Arrays class is defined to accept a Comparable[] array so that the compiler can complain if anyone ever calls sort with an array whose element type doesn't implement the Comparable interface. Sadly, that is not the case. Instead, the sort method accepts an Object[] array and uses a clumsy cast:

```java
// Approach used in the standard library--not recommended
if (((Comparable) a[i]).compareTo(a[j]) > 0)
{
    // rearrange a[i] and a[j]
    ...
}
```

If a[i] does not belong to a class that implements the Comparable interface, the virtual machine throws an exception.

Listing 6.1 presents the full code for sorting an array of instances of the class Employee (Listing 6.2). for sorting an employee array.

```
package interfaces;
import java.util.*;

/**
 * This program demonstrates the use of the Comparable interface.
 * @version 1.30 2004-02-27
 * @author Cay Horstmann
 */
public class EmployeeSortTest
{
    public static void main(String[] args)
    {
        Employee[] staff = new Employee[3];

        staff[0] = new Employee("Harry Hacker", 35000);
        staff[1] = new Employee("Carl Cracker", 75000);
        staff[2] = new Employee("Tony Tester", 38000);

        Arrays.sort(staff);

        // print out information about all Employee objects
        for (Employee e : staff)
            System.out.println("name=" + e.getName() + ",salary=" + e.getSalary());
    }
```

Listing 6.2 interfaces/Employee.java

```java
package interfaces;

public class Employee implements Comparable<Employee>
{
    private String name;
    private double salary;

    public Employee(String n, double s)
    {
        name = n;
        salary = s;
    }

    public String getName()
    {
        return name;
    }

    public double getSalary()
    {
        return salary;
    }

    public void raiseSalary(double byPercent)
    {
        double raise = salary * byPercent / 100;
        salary += raise;
    }

    /**
     * Compares employees by salary
     * @param other another Employee object
     * @return a negative value if this employee has a lower salary than
     *         otherObject, 0 if the salaries are the same, a positive value otherwise
     */
    public int compareTo(Employee other)
    {
        return Double.compare(salary, other.salary);
    }
}
```

*java.lang.Comparable<T>* 1.0

- **int compareTo(T other)**
  - compares this object with other and returns a negative integer if this object is less than other, zero if they are equal, and a positive integer otherwise.
java.util.Arrays 1.2

- static void sort(Object[] a)
  sorts the elements in the array a, using a tuned mergesort algorithm. All elements in
the array must belong to classes that implement the Comparable interface, and they
must all be comparable to each other.

java.lang.Integer 7

- static int compare(int x, int y)
  returns a negative integer if \( x < y \), zero if \( x \) and \( y \) are equal, and a positive integer
otherwise.

java.lang.Double 7

- static int compare(double x, double y)
  returns a negative integer if \( x < y \), zero if \( x \) and \( y \) are equal, and a positive integer
otherwise.

NOTE: According to the language standard: “The implementor must ensure
\( \text{sgn}(x.\text{compareTo}(y)) = -\text{sgn}(y.\text{compareTo}(x)) \) for all \( x \) and \( y \). (This implies that \( x.\text{compareTo}(y) \)
must throw an exception if \( y.\text{compareTo}(x) \) throws an exception.)” Here, \( \text{sgn} \) is the
\textit{sign} of a number: \( \text{sgn}(n) \) is –1 if \( n \) is negative, 0 if \( n \) equals 0, and 1 if \( n \) is positive.
In plain English, if you flip the parameters of \text{compareTo}, the sign (but not necessarily
the actual value) of the result must also flip.

As with the equals method, problems can arise when inheritance comes into play.
Since Manager extends Employee, it implements Comparable<Employee> and not
Comparable<Manager>. If Manager chooses to override compareTo, it must be prepared to
compare managers to employees. It can't simply cast an employee to a manager:

```java
class Manager extends Employee
{
    public int compareTo(Employee other)
    {
        Manager otherManager = (Manager) other; // NO
        
        . . .
    }
    . . .
}
```
That violates the “antisymmetry” rule. If $x$ is an Employee and $y$ is a Manager, then the call $x$.compareTo($y$) doesn’t throw an exception—it simply compares $x$ and $y$ as employees. But the reverse, $y$.compareTo($x$), throws a ClassCastException.

This is the same situation as with the equals method that we discussed in Chapter 5, and the remedy is the same. There are two distinct scenarios.

If subclasses have different notions of comparison, then you should outlaw comparison of objects that belong to different classes. Each compareTo method should start out with the test

```
if (getClass() != other.getClass()) throw new ClassCastException();
```

If there is a common algorithm for comparing subclass objects, simply provide a single compareTo method in the superclass and declare it as final.

For example, suppose that you want managers to be better than regular employees, regardless of the salary. What about other subclasses such as Executive and Secretary? If you need to establish a pecking order, supply a method such as rank in the Employee class. Have each subclass override rank, and implement a single compareTo method that takes the rank values into account.

### 6.1.1 Properties of Interfaces

Interfaces are not classes. In particular, you can never use the new operator to instantiate an interface:

```
x = new Comparable(. . .); // ERROR
```

However, even though you can’t construct interface objects, you can still declare interface variables.

```
Comparable x; // OK
```

An interface variable must refer to an object of a class that implements the interface:

```
x = new Employee(. . .); // OK provided Employee implements Comparable
```

Next, just as you use instanceof to check whether an object is of a specific class, you can use instanceof to check whether an object implements an interface:

```
if (anObject instanceof Comparable) {
...
```
Just as you can build hierarchies of classes, you can extend interfaces. This allows for multiple chains of interfaces that go from a greater degree of generality to a greater degree of specialization. For example, suppose you had an interface called Moveable.

```java
public interface Moveable
{
    void move(double x, double y);
}
```

Then, you could imagine an interface called Powered that extends it:

```java
public interface Powered extends Moveable
{
    double milesPerGallon();
}
```

Although you cannot put instance fields or static methods in an interface, you can supply constants in them. For example:

```java
public interface Powered extends Moveable
{
    double milesPerGallon();
    double SPEED_LIMIT = 95; // a public static final constant
}
```

Just as methods in an interface are automatically public, fields are always public static final.

**NOTE:** It is legal to tag interface methods as public, and fields as public static final. Some programmers do that, either out of habit or for greater clarity. However, the Java Language Specification recommends that the redundant keywords not be supplied, and we follow that recommendation.

Some interfaces define just constants and no methods. For example, the standard library contains an interface SwingConstants that defines constants NORTH, SOUTH, HORIZONTAL, and so on. Any class that chooses to implement the SwingConstants interface automatically inherits these constants. Its methods can simply refer to NORTH rather than the more cumbersome SwingConstants.NORTH. However, this use of interfaces seems rather degenerate, and we do not recommend it.

While each class can have only one superclass, classes can implement multiple interfaces. This gives you the maximum amount of flexibility in defining a class’s
behavior. For example, the Java programming language has an important interface built into it, called Cloneable. (We will discuss this interface in detail in the next section.) If your class implements Cloneable, the clone method in the Object class will make an exact copy of your class’s objects. Suppose, therefore, you want cloneability and comparability. Then you simply implement both interfaces.

class Employee implements Cloneable, Comparable

Use commas to separate the interfaces that describe the characteristics that you want to supply.

6.1.2 Interfaces and Abstract Classes

If you read the section about abstract classes in Chapter 5, you may wonder why the designers of the Java programming language bothered with introducing the concept of interfaces. Why can’t Comparable simply be an abstract class:

abstract class Comparable // why not?
{
    public abstract int compareTo(Object other);
}

The Employee class would then simply extend this abstract class and supply the compareTo method:

class Employee extends Comparable // why not?
{
    public int compareTo(Object other) { . . . }
}

There is, unfortunately, a major problem with using an abstract base class to express a generic property. A class can only extend a single class. Suppose that the Employee class already extends a different class, say, Person. Then it can’t extend a second class.

class Employee extends Person, Comparable // ERROR

But each class can implement as many interfaces as it likes:

class Employee extends Person implements Comparable // OK

Other programming languages, in particular C++, allow a class to have more than one superclass. This feature is called multiple inheritance. The designers of Java chose not to support multiple inheritance, because it makes the language either very complex (as in C++) or less efficient (as in Eiffel).

Instead, interfaces afford most of the benefits of multiple inheritance while avoiding the complexities and inefficiencies.
C++ NOTE: C++ has multiple inheritance and all the complications that come with it, such as virtual base classes, dominance rules, and transverse pointer casts. Few C++ programmers use multiple inheritance, and some say it should never be used. Other programmers recommend using multiple inheritance only for the "mix-in" style of inheritance. In the mix-in style, a primary base class describes the parent object, and additional base classes (the so-called mix-ins) may supply auxiliary characteristics. That style is similar to a Java class with a single base class and additional interfaces. However, in C++, mix-ins can add default behavior, whereas Java interfaces cannot.

6.2 Object Cloning

When you make a copy of a variable, the original and the copy are references to the same object. (See Figure 6.1.) This means a change to either variable also affects the other.

```java
Employee original = new Employee("John Public", 50000);
Employee copy = original;
copy.raiseSalary(10); // oops--also changed original
```

If you would like copy to be a new object that begins its life being identical to original but whose state can diverge over time, use the clone method.

```java
Employee copy = original.clone();
copy.raiseSalary(10); // OK--original unchanged
```

But it isn’t quite so simple. The clone method is a protected method of Object, which means that your code cannot simply call it. Only the Employee class can clone Employee objects. There is a reason for this restriction. Think about the way in which the Object class can implement clone. It knows nothing about the object at all, so it can make only a field-by-field copy. If all data fields in the object are numbers or other basic types, copying the fields is just fine. But if the object contains references to subobjects, then copying the field gives you another reference to the same subobject, so the original and the cloned objects still share some information.

To visualize that phenomenon, let’s consider the Employee class that was introduced in Chapter 4. Figure 6.2 shows what happens when you use the clone method of the Object class to clone such an Employee object. As you can see, the default cloning operation is “shallow”—it doesn’t clone objects that are referenced inside other objects.

Does it matter if the copy is shallow? It depends. If the subobject shared between the original and the shallow clone is immutable, then the sharing is safe. This certainly happens if the subobject belongs to an immutable class, such as String.
Alternatively, the subobject may simply remain constant throughout the lifetime of the object, with no mutators touching it and no methods yielding a reference to it.

Quite frequently, however, subobjects are mutable, and you must redefine the `clone` method to make a deep copy that clones the subobjects as well. In our example, the `hireDay` field is a `Date`, which is mutable.

For every class, you need to decide whether
1. The default `clone` method is good enough;
2. The default `clone` method can be patched up by calling `clone` on the mutable subobjects; and
3. `clone` should not be attempted.

The third option is actually the default. To choose either the first or the second option, a class must
1. Implement the `Cloneable` interface; and
2. Redefine the `clone` method with the `public` access modifier.

**NOTE:** The `clone` method is declared `protected` in the `Object` class, so that your code can't simply call `anObject.clone()`. But aren't protected methods accessible from any subclass, and isn't every class a subclass of `Object`? Fortunately, the rules for protected access are more subtle (see Chapter 5). A subclass can call a protected `clone` method only to clone its own objects. You must redefine `clone` to be public to allow objects to be cloned by any method.

In this case, the appearance of the `Cloneable` interface has nothing to do with the normal use of interfaces. In particular, it does not specify the `clone` method—that method is inherited from the `Object` class. The interface merely serves as a tag,
indicating that the class designer understands the cloning process. Objects are so paranoid about cloning that they generate a checked exception if an object requests cloning but does not implement that interface.

**NOTE:** The Cloneable interface is one of a handful of tagging interfaces that Java provides. (Some programmers call them marker interfaces.) Recall that the usual purpose of an interface such as Comparable is to ensure that a class implements a particular method or set of methods. A tagging interface has no methods; its only purpose is to allow the use of instanceof in a type inquiry:

```java
if (obj instanceof Cloneable) ...
```

We recommend that you do not use tagging interfaces in your own programs.

Even if the default (shallow copy) implementation of clone is adequate, you still need to implement the Cloneable interface, redefine clone to be public, and call super.clone(). Here is an example:

```java
class Employee implements Cloneable {
    // raise visibility level to public, change return type
    public Employee clone() throws CloneNotSupportedException {
        return (Employee) super.clone();
    }
    ...
}
```

**NOTE:** Before Java SE 5.0, the clone method always had return type Object. Nowadays, the covariant return type feature lets you specify the correct return type for your clone methods.

The clone method that you just saw adds no functionality to the shallow copy provided by Object.clone. It merely makes the method public. To make a deep copy, you have to work harder and clone the mutable instance fields.

Here is an example of a clone method that creates a deep copy:

```java
class Employee implements Cloneable {
    public Employee clone() throws CloneNotSupportedException {
        // call Object.clone()
        Employee cloned = (Employee) super.clone();
    }
}
```
The clone method of the Object class threatens to throw a CloneNotSupportedException—it does that whenever clone is invoked on an object whose class does not implement the Cloneable interface. Of course, the Employee and Date classes implement the Cloneable interface, so the exception won’t be thrown. However, the compiler does not know that. Therefore, we declared the exception:

```java
public Employee clone() throws CloneNotSupportedException
```

Would it be better to catch the exception instead?

```java
public Employee clone()
{
    try
    {
        return (Employee) super.clone();
    }
    catch (CloneNotSupportedException e) { return null; }
    // this won't happen, since we are Cloneable
}
```

This is appropriate for final classes. Otherwise, it is a good idea to leave the throws specifier in place. That gives subclasses the option of throwing a CloneNotSupportedException if they can’t support cloning.

You have to be careful about cloning of subclasses. For example, once you have defined the clone method for the Employee class, anyone can use it to clone Manager objects. Can the Employee clone method do the job? It depends on the fields of the Manager class. In our case, there is no problem because the bonus field has primitive type. But Manager might have acquired fields that require a deep copy or are not cloneable. There is no guarantee that the implementor of the subclass has fixed clone to do the right thing. For that reason, the clone method is declared as protected in the Object class. But you don’t have that luxury if you want users of your classes to invoke clone.

Should you implement clone in your own classes? If your clients need to make deep copies, then you probably should. Some authors feel that you should avoid clone altogether and instead implement another method for the same purpose. We agree that clone is rather awkward, but you’ll run into the same issues if you shift the responsibility to another method. At any rate, cloning is less common
than you may think. Less than 5 percent of the classes in the standard library implement clone.

The program in Listing 6.3 clones an instance of the class Employee (Listing 6.4), then invokes two mutators. The raiseSalary method changes the value of the salary field, whereas the setHireDay method changes the state of the hireDay field. Neither mutation affects the original object because clone has been defined to make a deep copy.

Listing 6.3  clone/CloneTest.java

```java
package clone;

/**
 * This program demonstrates cloning.
 * @version 1.10 2002-07-01
 * @author Cay Horstmann
 */
public class CloneTest
{
    public static void main(String[] args)
    {
        try
        {
            Employee original = new Employee("John Q. Public", 50000);
            original.setHireDay(2000, 1, 1);
            Employee copy = original.clone();
            copy.raiseSalary(10);
            copy.setHireDay(2002, 12, 31);
            System.out.println("original=" + original);
            System.out.println("copy=" + copy);
        }
        catch (CloneNotSupportedException e)
        {
            e.printStackTrace();
        }
    }
}
```

Listing 6.4  clone/Employee.java

```java
package clone;

import java.util.Date;
import java.util.GregorianCalendar;
```
public class Employee implements Cloneable {
    private String name;
    private double salary;
    private Date hireDay;

    public Employee(String n, double s) {
        name = n;
        salary = s;
        hireDay = new Date();
    }

    public Employee clone() throws CloneNotSupportedException {
        // call Object.clone()
        Employee cloned = (Employee) super.clone();

        // clone mutable fields
        cloned.hireDay = (Date) hireDay.clone();

        return cloned;
    }

    /**
     * Set the hire day to a given date.
     * @param year the year of the hire day
     * @param month the month of the hire day
     * @param day the day of the hire day
     */
    public void setHireDay(int year, int month, int day) {
        Date newHireDay = new GregorianCalendar(year, month - 1, day).getTime();

        // Example of instance field mutation
        hireDay.setTime(newHireDay.getTime());
    }

    public void raiseSalary(double byPercent) {
        double raise = salary * byPercent / 100;
        salary += raise;
    }

    public String toString() {
        return "Employee[ name=\" + name + ", salary=\" + salary + ", hireDay=\" + hireDay + "]";
    }
}

6.2 Object Cloning
NOTE: All array types have a `clone` method that is public, not protected. You can use it to make a new array that contains copies of all elements. For example:

```java
int[] luckyNumbers = { 2, 3, 5, 7, 11, 13 };
int[] cloned = luckyNumbers.clone();
```

NOTE: Chapter 1 of Volume II shows an alternate mechanism for cloning objects, using the object serialization feature of Java. That mechanism is easy to implement and safe, but not very efficient.

### 6.3 Interfaces and Callbacks

A common pattern in programming is the *callback* pattern. In this pattern, you want to specify the action that should occur whenever a particular event happens. For example, you may want a particular action to occur when a button is clicked or a menu item is selected. However, as you have not yet seen how to implement user interfaces, we will consider a similar but simpler situation.

The `javax.swing` package contains a `Timer` class that is useful if you want to be notified whenever a time interval has elapsed. For example, if a part of your program contains a clock, then you can ask to be notified every second so that you can update the clock face.

When you construct a timer, you set the time interval and you tell it what it should do whenever the time interval has elapsed.

How do you tell the timer what it should do? In many programming languages, you supply the name of a function that the timer should call periodically. However, the classes in the Java standard library take an object-oriented approach. You pass an object of some class. The timer then calls one of the methods on that object. Passing an object is more flexible than passing a function because the object can carry additional information.

Of course, the timer needs to know what method to call. The timer requires that you specify an object of a class that implements the `ActionListener` interface of the `java.awt.event` package. Here is that interface:

```java
public interface ActionListener
{
    void actionPerformed(ActionEvent event);
}
```

The timer calls the `actionPerformed` method when the time interval has expired.
Suppose you want to print a message “At the tone, the time is . . .”, followed by a beep, once every 10 seconds. You would define a class that implements the ActionListener interface. You would then place whatever statements you want to have executed inside the actionPerformed method.

```java
class TimePrinter implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        Date now = new Date();
        System.out.println("At the tone, the time is "+ now);
        Toolkit.getDefaultToolkit().beep();
    }
}
```

Note the ActionEvent parameter of the actionPerformed method. This parameter gives information about the event, such as the source object that generated it—see Chapter 8 for more information. However, detailed information about the event is not important in this program, and you can safely ignore the parameter.

Next, you construct an object of this class and pass it to the Timer constructor.

```java
ActionListener listener = new TimePrinter();
Timer t = new Timer(10000, listener);
```

The first parameter of the Timer constructor is the time interval that must elapse between notifications, measured in milliseconds. We want to be notified every 10 seconds. The second parameter is the listener object.

Finally, you start the timer.

```java
t.start();
```

Every 10 seconds, a message like

```
At the tone, the time is Thu Apr 13 23:29:08 PDT 2000
```

is displayed, followed by a beep.

Listing 6.5 puts the timer and its action listener to work. After the timer is started, the program puts up a message dialog and waits for the user to click the Ok button to stop. While the program waits for the user, the current time is displayed at 10-second intervals.
Be patient when running the program. The “Quit program?” dialog box appears right away, but the first timer message is displayed after 10 seconds.

Note that the program imports the javax.swing.Timer class by name, in addition to importing javax.swing.* and java.util.*. This breaks the ambiguity between javax.swing.Timer and java.util.Timer, an unrelated class for scheduling background tasks.

Listing 6.5 timer/TimerTest.java

```java
1  package timer;
2  /**
3     @version 1.00 2000-04-13
4     @author Cay Horstmann
5  */
6
7  import java.awt.*;
8  import java.awt.event.*;
9  import java.util.*;
10  import javax.swing.*;
11  import javax.swing.Timer;
12  // to resolve conflict with java.util.Timer
13
14  public class TimerTest
15  {
16     public static void main(String[] args)
17     {
18        ActionListener listener = new TimePrinter();
19
20        // construct a timer that calls the listener
21        // once every 10 seconds
22        Timer t = new Timer(10000, listener);
23        t.start();
24
25        JOptionPane.showMessageDialog(null, "Quit program?");
26        System.exit(0);
27     }
28  }
29
30  class TimePrinter implements ActionListener
31  {
32     public void actionPerformed(ActionEvent event)
33     {
34        Date now = new Date();
35        System.out.println("At the tone, the time is " + now);
36        Toolkit.getDefaultToolkit().beep();
37     }
38  }
```
### javax.swing.JOptionPane 1.2

- static void showMessageDialog(Component parent, Object message)
  
displays a dialog box with a message prompt and an Ok button. The dialog is centered over the parent component. If parent is null, the dialog is centered on the screen.

### javax.swing.Timer 1.2

- Timer(int interval, ActionListener listener)
  
constructs a timer that notifies listener whenever interval milliseconds have elapsed.
- void start()
  
starts the timer. Once started, the timer calls actionPerformed on its listeners.
- void stop()
  
stops the timer. Once stopped, the timer no longer calls actionPerformed on its listeners.

### java.awt.Toolkit 1.0

- static Toolkit getDefaultToolkit()
  
gets the default toolkit. A toolkit contains information about the GUI environment.
- void beep()
  
emits a beep sound.

### 6.4 Inner Classes

An **inner class** is a class that is defined inside another class. Why would you want to do that? There are three reasons:

- Inner class methods can access the data from the scope in which they are defined—including the data that would otherwise be private.
- Inner classes can be hidden from other classes in the same package.
- *Anonymous* inner classes are handy when you want to define callbacks without writing a lot of code.

We will break up this rather complex topic into several steps.

1. Starting on page 307, you will see a simple inner class that accesses an instance field of its outer class.
2. On page 311, we cover the special syntax rules for inner classes.
3. Starting on page 312, we peek inside inner classes to see how they are translated into regular classes. Squeamish readers may want to skip that section.

4. Starting on page 315, we discuss local inner classes that can access local variables of the enclosing scope.

5. Starting on page 318, we introduce anonymous inner classes and show how they are commonly used to implement callbacks.

6. Finally, starting on page 322, you will see how static inner classes can be used for nested helper classes.

**C++ NOTE:**

C++ has *nested classes.* A nested class is contained inside the scope of the enclosing class. Here is a typical example: A linked list class defines a class to hold the links, and a class to define an iterator position.

```cpp
class LinkedList
{
    public:
        class Iterator // a nested class
        {
            public:
                void insert(int x);
                int erase();
                . . .
            }
        . . .
    private:
        class Link // a nested class
        {
            public:
                Link* next;
                int data;
            }
        . . .
};
```

The nesting is a relationship between classes, not objects. A LinkedList object does not have subobjects of type Iterator or Link.

There are two benefits: *name control* and *access control.* The name Iterator is nested inside the LinkedList class, so it is known externally as LinkedList::Iterator and cannot conflict with another class called Iterator. In Java, this benefit is not as important because Java *packages* give the same kind of name control. Note that the Link class is in the *private* part of the LinkedList class. It is completely
hidden from all other code. For that reason, it is safe to make its data fields public. They can be accessed by the methods of the LinkedList class (which has a legitimate need to access them), and they are not visible elsewhere. In Java, this kind of control was not possible until inner classes were introduced.

However, the Java inner classes have an additional feature that makes them richer and more useful than nested classes in C++. An object that comes from an inner class has an implicit reference to the outer class object that instantiated it. Through this pointer, it gains access to the total state of the outer object. You will see the details of the Java mechanism later in this chapter.

In Java, static inner classes do not have this added pointer. They are the Java analog to nested classes in C++.

### 6.4.1 Use of an Inner Class to Access Object State

The syntax for inner classes is rather complex. For that reason, we present a simple but somewhat artificial example to demonstrate the use of inner classes. We refactor the TimerTest example and extract a TalkingClock class. A talking clock is constructed with two parameters: the interval between announcements and a flag to turn beeps on or off.

```java
public class TalkingClock {
    private int interval;
    private boolean beep;

    public TalkingClock(int interval, boolean beep) { ... }
    public void start() { ... }

    public class TimePrinter implements ActionListener {
        // an inner class
        { ... }
    }
}
```

Note that the TimePrinter class is now located inside the TalkingClock class. This does not mean that every TalkingClock has a TimePrinter instance field. As you will see, the TimePrinter objects are constructed by methods of the TalkingClock class.

Here is the TimePrinter class in greater detail. Note that the actionPerformed method checks the beep flag before emitting a beep.
public class TimePrinter implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {
        Date now = new Date();
        System.out.println("At the tone, the time is " + now);
        if (beep) Toolkit.getDefaultToolkit().beep();
    }
}

Something surprising is going on. The TimePrinter class has no instance field or variable named beep. Instead, beep refers to the field of the TalkingClock object that created this TimePrinter. This is quite innovative. Traditionally, a method could refer to the data fields of the object invoking the method. An inner class method gets to access both its own data fields and those of the outer object creating it.

For this to work, an object of an inner class always gets an implicit reference to the object that created it. (See Figure 6.3.)

This reference is invisible in the definition of the inner class. However, to illuminate the concept, let us call the reference to the outer object outer. Then, the actionPerformed method is equivalent to the following:

Figure 6.3 An inner class object has a reference to an outer class object
public void actionPerformed(ActionEvent event) {
    Date now = new Date();
    System.out.println("At the tone, the time is " + now);
    if (outer.beep) Toolkit.getDefaultToolkit().beep();
}

The outer class reference is set in the constructor. The compiler modifies all inner class constructors, adding a parameter for the outer class reference. The TimePrinter class defines no constructors; therefore, the compiler synthesizes a no-argument constructor, generating code like this:

    public TimePrinter(TalkingClock clock) // automatically generated code
    {
        outer = clock;
    }

Again, please note, outer is not a Java keyword. We just use it to illustrate the mechanism involved in an inner class.

When a TimePrinter object is constructed in the start method, the compiler passes the this reference to the current talking clock into the constructor:

    ActionListener listener = new TimePrinter(this); // parameter automatically added

Listing 6.6 shows the complete program that tests the inner class. Have another look at the access control. Had the TimePrinter class been a regular class, it would have needed to access the beep flag through a public method of the TalkingClock class. Using an inner class is an improvement. There is no need to provide accessors that are of interest only to one other class.

NOTE: We could have declared the TimePrinter class as private. Then only TalkingClock methods would be able to construct TimePrinter objects. Only inner classes can be private. Regular classes always have either package or public visibility.

Listing 6.6 innerClass/InnerClassTest.java

1  package innerClass;
  
2  import java.awt.*;
3  import java.awt.event.*;
4  import java.util.*;
5  import java.util.*;
6  import javax.swing.*;
7  import javax.swing.Timer;
(Continues)
Listing 6.6 (Continued)

/*
 * This program demonstrates the use of inner classes.
 * @version 1.10 2004-02-27
 * @author Cay Horstmann
 */
public class InnerClassTest {
    public static void main(String[] args) {
        TalkingClock clock = new TalkingClock(1000, true);
        clock.start();
        // keep program running until user selects "Ok"
        JOptionPane.showMessageDialog(null, "Quit program?");
        System.exit(0);
    }
}

/**
 * A clock that prints the time in regular intervals.
 */
class TalkingClock {
    private int interval;
    private boolean beep;

    /**
     * Constructs a talking clock
     * @param interval the interval between messages (in milliseconds)
     * @param beep true if the clock should beep
     */
    public TalkingClock(int interval, boolean beep) {
        this.interval = interval;
        this.beep = beep;
    }

    /**
     * Starts the clock.
     */
    public void start() {
        ActionListener listener = new TimePrinter();
        Timer t = new Timer(interval, listener);
        t.start();
    }
public class TimePrinter implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {
        Date now = new Date();
        System.out.println("At the tone, the time is "+ now);
        if (beep) Toolkit.getDefaultToolkit().beep();
    }
}

6.4.2 Special Syntax Rules for Inner Classes

In the preceding section, we explained the outer class reference of an inner class by calling it outer. Actually, the proper syntax for the outer reference is a bit more complex. The expression

    OuterClass.this

denotes the outer class reference. For example, you can write the actionPerformed method of the TimePrinter inner class as

    public void actionPerformed(ActionEvent event)
    {
        ...
        if (TalkingClock.this.beep) Toolkit.getDefaultToolkit().beep();
    }

Conversely, you can write the inner object constructor more explicitly, using the syntax

    outerObject.newInstance( construction parameters )

For example:

    ActionListener listener = this.newInstance(TimePrinter);

Here, the outer class reference of the newly constructed TimePrinter object is set to the this reference of the method that creates the inner class object. This is the most common case. As always, the this. qualifier is redundant. However, it is also possible to set the outer class reference to another object by explicitly naming it. For example, since TimePrinter is a public inner class, you can construct a TimePrinter for any talking clock:

    TalkingClock jabberer = new TalkingClock(1000, true);
    TalkingClock.TimePrinter listener = jabberer.newInstance();

Note that you refer to an inner class as

    OuterClass .InnerClass

when it occurs outside the scope of the outer class.
6.4.3 Are Inner Classes Useful? Actually Necessary? Secure?

When inner classes were added to the Java language in Java 1.1, many programmers considered them a major new feature that was out of character with the Java philosophy of being simpler than C++. The inner class syntax is undeniably complex. (It gets more complex as we study anonymous inner classes later in this chapter.) It is not obvious how inner classes interact with other features of the language, such as access control and security.

By adding a feature that was elegant and interesting rather than needed, has Java started down the road to ruin which has afflicted so many other languages?

While we won’t try to answer this question completely, it is worth noting that inner classes are a phenomenon of the compiler, not the virtual machine. Inner classes are translated into regular class files with $ (dollar signs) delimiting outer and inner class names, and the virtual machine does not have any special knowledge about them.

For example, the TimePrinter class inside the TalkingClock class is translated to a class file TalkingClock$TimePrinter.class. To see this at work, try the following experiment: run the ReflectionTest program of Chapter 5, and give it the class TalkingClock$TimePrinter to reflect upon. Alternatively, simply use the javap utility:

```
javap -private ClassName
```

NOTE: If you use UNIX, remember to escape the $ character if you supply the class name on the command line. That is, run the ReflectionTest or javap program as

```
java reflection.ReflectionTest innerClass.TalkingClock$TimePrinter
```

or

```
javap -private innerClass.TalkingClock$TimePrinter
```

You will get the following printout:

```
public class TalkingClock$TimePrinter
{
    public TalkingClock$TimePrinter(TalkingClock);
    public void actionPerformed(java.awt.event.ActionEvent);
    final TalkingClock this$0;
}
```
You can plainly see that the compiler has generated an additional instance field, this$0, for the reference to the outer class. (The name this$0 is synthesized by the compiler—you cannot refer to it in your code.) You can also see the TalkingClock parameter for the constructor.

If the compiler can automatically do this transformation, couldn’t you simply program the same mechanism by hand? Let’s try it. We would make TimePrinter a regular class, outside the TalkingClock class. When constructing a TimePrinter object, we pass it the this reference of the object that is creating it.

class TalkingClock
{
    ...
    public void start()
    {
        ActionListener listener = new TimePrinter(this);
        Timer t = new Timer(interval, listener);
        t.start();
    }
}

class TimePrinter implements ActionListener
{
    private TalkingClock outer;
    ...
    public TimePrinter(TalkingClock clock)
    {
        outer = clock;
    }
}

Now let us look at the actionPerformed method. It needs to access outer.beep.

    if (outer.beep) . . . // ERROR

Here we run into a problem. The inner class can access the private data of the outer class, but our external TimePrinter class cannot.

Thus, inner classes are genuinely more powerful than regular classes because they have more access privileges.

You may well wonder how inner classes manage to acquire those added access privileges, if they are translated to regular classes with funny names—the virtual machine knows nothing at all about them. To solve this mystery, let’s again use the ReflectionTest program to spy on the TalkingClock class:
class TalkingClock
{
    private int interval;
    private boolean beep;

    public TalkingClock(int, boolean);
    static boolean access$0(TalkingClock);
    public void start();
}

Notice the static access$0 method that the compiler added to the outer class. It returns the beep field of the object that is passed as a parameter. (The method name might be slightly different, such as access$000, depending on your compiler.)

The inner class methods call that method. The statement

    if (beep)

in the actionPerformed method of the TimePrinter class effectively makes the following call:

    if (access$0(outer));

Is this a security risk? You bet it is. It is an easy matter for someone else to invoke the access$0 method to read the private beep field. Of course, access$0 is not a legal name for a Java method. However, hackers who are familiar with the structure of class files can easily produce a class file with virtual machine instructions to call that method, for example, by using a hex editor. Since the secret access methods have package visibility, the attack code would need to be placed inside the same package as the class under attack.

To summarize, if an inner class accesses a private data field, then it is possible to access that data field through other classes added to the package of the outer class, but to do so requires skill and determination. A programmer cannot accidentally obtain access but must intentionally build or modify a class file for that purpose.

NOTE: The synthesized constructors and methods can get quite convoluted. (Skip this note if you are squeamish.) Suppose we turn TimePrinter into a private inner class. There are no private classes in the virtual machine, so the compiler produces the next best thing: a package-visible class with a private constructor

    private TalkingClock$TimePrinter(TalkingClock);

Of course, nobody can call that constructor, so there is a second package-visible constructor
TalkingClock$TimePrinter(TalkingClock, TalkingClock$1);

that calls the first one.

The compiler translates the constructor call in the `start` method of the `TalkingClock` class to

```java
new TalkingClock$TimePrinter(this, null)
```

### 6.4.4 Local Inner Classes

If you look carefully at the code of the `TalkingClock` example, you will find that you need the name of the type `TimePrinter` only once: when you create an object of that type in the `start` method.

In a situation like this, you can define the class *locally in a single method*.

```java
public void start()
{
  class TimePrinter implements ActionListener
  {
    public void actionPerformed(ActionEvent event)
    {
      Date now = new Date();
      System.out.println("At the tone, the time is \" + now);
      if (beep) Toolkit.getDefaultToolkit().beep();
    }
  }

  ActionListener listener = new TimePrinter();
  Timer t = new Timer(interval, listener);
  t.start();
}
```

Local classes are never declared with an access specifier (that is, `public` or `private`). Their scope is always restricted to the block in which they are declared.

Local classes have one great advantage: They are completely hidden from the outside world—not even other code in the `TalkingClock` class can access them. No method except `start` has any knowledge of the `TimePrinter` class.

### 6.4.5 Accessing final Variables from Outer Methods

Local classes have another advantage over other inner classes. Not only can they access the fields of their outer classes; they can even access local variables! However, those local variables must be declared `final`. Here is a typical example.
Let’s move the interval and beep parameters from theTalkingClock constructor to the start method.

```java
public void start(int interval, final boolean beep)
{
    class TimePrinter implements ActionListener
    {
        public void actionPerformed(ActionEvent event)
        {
            Date now = new Date();
            System.out.println("At the tone, the time is " + now);
            if (beep) Toolkit.getDefaultToolkit().beep();
        }
    }
    ActionListener listener = new TimePrinter();
    Timer t = new Timer(interval, listener);
    t.start();
}
```

Note that the TalkingClock class no longer needs to store a beep instance field. It simply refers to the beep parameter variable of the start method.

Maybe this should not be so surprising. The line

```java
if (beep) . . .
```

is, after all, ultimately inside the start method, so why shouldn’t it have access to the value of the beep variable?

To see why there is a subtle issue here, let’s consider the flow of control more closely.

1. The start method is called.
2. The object variable listener is initialized by a call to the constructor of the inner class TimePrinter.
3. The listener reference is passed to the Timer constructor, the timer is started, and the start method exits. At this point, the beep parameter variable of the start method no longer exists.
4. A second later, the actionPerformed method executes if (beep) . . .

For the code in the actionPerformed method to work, the TimePrinter class must have copied the beep field, as a local variable of the start method, before the beep parameter value went away. That is indeed exactly what happens. In our example, the compiler synthesizes the name TalkingClock$1TimePrinter for the local inner class. If you use the ReflectionTest program again to spy on the TalkingClock$1TimePrinter class, you will get the following output:
class TalkingClock$1TimePrinter
{
    TalkingClock$1TimePrinter(TalkingClock, boolean);

    public void actionPerformed(java.awt.event.ActionEvent);

    final boolean val$beep;
    final TalkingClock this$0;
}

Note the boolean parameter to the constructor and the val$beep instance variable. When an object is created, the value beep is passed into the constructor and stored in the val$beep field. The compiler detects access of local variables, makes matching instance fields for each one of them, and copies the local variables into the constructor so that the instance fields can be initialized.

From the programmer’s point of view, local variable access is quite pleasant. It makes your inner classes simpler by reducing the instance fields that you need to program explicitly.

As we already mentioned, the methods of a local class can refer only to local variables that are declared final. For that reason, the beep parameter was declared final in our example. A local variable that is declared final cannot be modified after it has been initialized. Thus, it is guaranteed that the local variable and the copy that is made inside the local class will always have the same value.

NOTE: You have seen final variables used for constants, such as

    public static final double SPEED_LIMIT = 55;

The final keyword can be applied to local variables, instance variables, and static variables. In all cases it means the same thing: You can assign to this variable once after it has been created. Afterwards, you cannot change the value—it is final.

However, you don't have to initialize a final variable when you define it. For example, the final parameter variable beep is initialized once after its creation, when the start method is called. (If the method is called multiple times, each call has its own newly created beep parameter.) The val$beep instance variable that you can see in the TalkingClock$1TimePrinter inner class is set once, in the inner class constructor. A final variable that isn't initialized when it is defined is often called a blank final variable.
The final restriction is somewhat inconvenient. Suppose, for example, that you want to update a counter in the enclosing scope. Here, we want to count how often the `compareTo` method is called during sorting:

```java
int counter = 0;
Date[] dates = new Date[100];
for (int i = 0; i < dates.length; i++)
    dates[i] = new Date()
    {
        public int compareTo(Date other)
        {
            counter++; // ERROR
            return super.compareTo(other);
        }
    };
Arrays.sort(dates);
System.out.println(counter + " comparisons.");
```

You can’t declare `counter` as `final` because you clearly need to update it. You can’t replace it with an `Integer` because `Integer` objects are immutable. The remedy is to use an array of length 1:

```java
final int[] counter = new int[1];
for (int i = 0; i < dates.length; i++)
    dates[i] = new Date()
    {
        public int compareTo(Date other)
        {
            counter[0]++;
            return super.compareTo(other);
        }
    };
Arrays.sort(dates);
System.out.println(counter[0] + " comparisons.");
```

(The array variable is still declared as `final`, but that merely means that you can’t have it refer to a different array. You are free to mutate the array elements.)

When inner classes were first invented, the prototype compiler automatically made this transformation for all local variables that were modified in the inner class. However, some programmers were fearful of having the compiler produce heap objects behind their backs, and the `final` restriction was adopted instead. It is possible that a future version of the Java language will revise this decision.

### 6.4.6 Anonymous Inner Classes

When using local inner classes, you can often go a step further. If you want to make only a single object of this class, you don’t even need to give the class a name. Such a class is called an **anonymous inner class**.
public void start(int interval, final boolean beep) {
    ActionListener listener = new ActionListener() {
        public void actionPerformed(ActionEvent event) {
            Date now = new Date();
            System.out.println("At the tone, the time is " + now);
            if (beep) Toolkit.getDefaultToolkit().beep();
        }
    };
    Timer t = new Timer(interval, listener);
    t.start();
}

This syntax is very cryptic indeed. What it means is this: Create a new object of a class that implements the ActionListener interface, where the required method actionPerformed is the one defined inside the braces {}.

In general, the syntax is

    new SuperType(construction parameters) {
        inner class methods and data
    }

Here, SuperType can be an interface, such as ActionListener; then, the inner class implements that interface. SuperType can also be a class; then, the inner class extends that class.

An anonymous inner class cannot have constructors because the name of a constructor must be the same as the name of a class, and the class has no name. Instead, the construction parameters are given to the superclass constructor. In particular, whenever an inner class implements an interface, it cannot have any construction parameters. Nevertheless, you must supply a set of parentheses as in

    new InterfaceType() {
        methods and data
    }

You have to look carefully to see the difference between the construction of a new object of a class and the construction of an object of an anonymous inner class extending that class.

Person queen = new Person("Mary");
// a Person object
Person count = new Person("Dracula") { ...};
// an object of an inner class extending Person
If the closing parenthesis of the construction parameter list is followed by an opening brace, then an anonymous inner class is being defined.

Are anonymous inner classes a great idea or are they a great way of writing obfuscated code? Probably a bit of both. When the code for an inner class is short, just a few lines of simple code, then anonymous inner classes can save typing time—but it is exactly such time-saving features that lead you down the slippery slope to “Obfuscated Java Code Contests.”

Listing 6.7 contains the complete source code for the talking clock program with an anonymous inner class. If you compare this program with Listing 6.6, you will find that in this case, the solution with the anonymous inner class is quite a bit shorter, and, hopefully, with a bit of practice, as easy to comprehend.

```
package anonymousInnerClass;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.Timer;

public class AnonymousInnerClassTest
{
    public static void main(String[] args)
    {
        TalkingClock clock = new TalkingClock();
        clock.start(1000, true);

        // keep program running until user selects "Ok"
        JOptionPane.showMessageDialog(null, "Quit program?");
        System.exit(0);
    }

    /**
     * This program demonstrates anonymous inner classes.
     * @version 1.10 2004-02-27
     * @author Cay Horstmann
     */
    public class AnonymousInnerClassTest
    { /* A clock that prints the time in regular intervals. */
```

Chapter 6 Interfaces and Inner Classes
class TalkingClock
{
    /**
     * Starts the clock.
     * @param interval the interval between messages (in milliseconds)
     * @param beep true if the clock should beep
     */
    public void start(int interval, final boolean beep)
    {
        ActionListener listener = new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                Date now = new Date();
                System.out.println("At the tone, the time is " + now);
                if (beep) Toolkit.getDefaultToolkit().beep();
            }
        };
        Timer t = new Timer(interval, listener);
        t.start();
    }
}

NOTE: The following trick, called double brace initialization, takes advantage of the inner class syntax. Suppose you want to construct an array list and pass it to a method:

    ArrayList<String> friends = new ArrayList<>();
    friends.add("Harry");
    friends.add("Tony");
    invite(friends);

If you don't need the array list again, it would be nice to make it anonymous. But then how can you add the elements? Here is how:

    invite(new ArrayList<String>() {{
        add("Harry");
        add("Tony");
    }})

Note the double braces. The outer braces make an anonymous subclass of ArrayList. The inner braces are an object construction block (see Chapter 4).

CAUTION: It is often convenient to make an anonymous subclass that is almost, but not quite, like its superclass. But you need to be careful with the equals method. In Chapter 5, we recommended that your equals methods use a test

    if (getClass() != other.getClass()) return false;

An anonymous subclass will fail this test.
TIP: When you produce logging or debugging messages, you often want to include the name of the current class, such as

```
System.err.println("Something awful happened in " + getClass());
```

But that fails in a static method. After all, the call to `getClass()` calls `this.getClass()`, and a static method has no `this`. Use the following expression instead:

```
new Object(){}.getClass().getEnclosingClass() // gets class of static method
```

Here, `new Object(){}` makes an anonymous object of an anonymous subclass of `Object`, and `getEnclosingClass` gets its enclosing class—that is, the class containing the static method.

### 6.4.7 Static Inner Classes

Occasionally, you may want to use an inner class simply to hide one class inside another—but you don’t need the inner class to have a reference to the outer class object. You can suppress the generation of that reference by declaring the inner class static.

Here is a typical example of where you would want to do this. Consider the task of computing the minimum and maximum value in an array. Of course, you write one method to compute the minimum and another method to compute the maximum. When you call both methods, the array is traversed twice. It would be more efficient to traverse the array only once, computing both the minimum and the maximum simultaneously.

```
double min = Double.MAX_VALUE;
double max = Double.MIN_VALUE;
for (double v : values)
{
    if (min > v) min = v;
    if (max < v) max = v;
}
```

However, the method must return two numbers. We can achieve that by defining a class `Pair` that holds two values:

```
class Pair
{
    private double first;
    private double second;

    public Pair(double f, double s)
    {
        first = f;
        second = s;
    }
```
The `minmax` method can then return an object of type `Pair`:

```java
class ArrayAlg {
    public static Pair minmax(double[] values) {
        ... 
        return new Pair(min, max);
    }
}
```

The caller of the method uses the `getFirst` and `getSecond` methods to retrieve the answers:

```java
Pair p = ArrayAlg.minmax(d);
System.out.println("min = " + p.getFirst());
System.out.println("max = " + p.getSecond());
```

Of course, the name `Pair` is an exceedingly common name, and in a large project, it is quite possible that some other programmer had the same bright idea—but made a `Pair` class that contains a pair of strings. We can solve this potential name clash by making `Pair` a public inner class inside `ArrayAlg`. Then the class will be known to the public as `ArrayAlg.Pair`:

```java
ArrayAlg.Pair p = ArrayAlg.minmax(d);
```

However, unlike the inner classes that we used in previous examples, we do not want to have a reference to any other object inside a `Pair` object. That reference can be suppressed by declaring the inner class `static`:

```java
class ArrayAlg {
    public static class Pair {
    ...
    }
    ...
}
```

Of course, only inner classes can be declared static. A static inner class is exactly like any other inner class, except that an object of a static inner class does not have a reference to the outer class object that generated it. In our example, we must use a static inner class because the inner class object is constructed inside a static method:
public static Pair minmax(double[] d) {
    ...
    return new Pair(min, max);
}

Had the Pair class not been declared as static, the compiler would have complained that there was no implicit object of type ArrayAlg available to initialize the inner class object.

**NOTE:** Use a static inner class whenever the inner class does not need to access an outer class object. Some programmers use the term *nested class* to describe static inner classes.

**NOTE:** Inner classes that are declared inside an interface are automatically static and public.

Listing 6.8 contains the complete source code of the ArrayAlg class and the nested Pair class.

```
package staticInnerClass;

/**
   * This program demonstrates the use of static inner classes.
   * @version 1.01 2004-02-27
   * @author Cay Horstmann
   */
public class StaticInnerClassTest {
    public static void main(String[] args) {
        double[] d = new double[20];
        for (int i = 0; i < d.length; i++)
            d[i] = 100 * Math.random();
        ArrayAlg.Pair p = ArrayAlg.minmax(d);
        System.out.println("min = " + p.getFirst());
        System.out.println("max = " + p.getSecond());
    }
}
```
class ArrayAlg {
  
  /**
   * A pair of floating-point numbers
   */
  public static class Pair {
    
    private double first;
    private double second;

    /**
     * Constructs a pair from two floating-point numbers
     * @param f the first number
     * @param s the second number
     */
    public Pair(double f, double s) {
      first = f;
      second = s;
    }

    /**
     * Returns the first number of the pair
     * @return the first number
     */
    public double getFirst() {
      return first;
    }

    /**
     * Returns the second number of the pair
     * @return the second number
     */
    public double getSecond() {
      return second;
    }
  }

  /**
   * Computes both the minimum and the maximum of an array
   * @param values an array of floating-point numbers
   * @return a pair whose first element is the minimum and whose second element
   * is the maximum
   */

  (Continues)
public static Pair minmax(double[] values) {
    double min = Double.MAX_VALUE;
    double max = Double.MIN_VALUE;
    for (double v : values) {
        if (min > v) min = v;
        if (max < v) max = v;
    }
    return new Pair(min, max);
}
However, you cannot define new code for these methods at runtime. Instead, you must supply an **invocation handler**. An invocation handler is an object of any class that implements the `InvocationHandler` interface. That interface has a single method:

```
Object invoke(Object proxy, Method method, Object[] args)
```

Whenever a method is called on the proxy object, the `invoke` method of the invocation handler gets called, with the `Method` object and parameters of the original call. The invocation handler must then figure out how to handle the call.

To create a proxy object, use the `newProxyInstance` method of the `Proxy` class. The method has three parameters:

- A **class loader**. As part of the Java security model, different class loaders can be used for system classes, classes that are downloaded from the Internet, and so on. We will discuss class loaders in Chapter 9 of Volume II. For now, we specify `null` to use the default class loader.
- An array of `Class` objects, one for each interface to be implemented.
- An invocation handler.

There are two remaining questions. How do we define the handler? And what can we do with the resulting proxy object? The answers depend, of course, on the problem that we want to solve with the proxy mechanism. Proxies can be used for many purposes, such as

- Routing method calls to remote servers
- Associating user interface events with actions in a running program
- Tracing method calls for debugging purposes

In our example program, we use proxies and invocation handlers to trace method calls. We define a `TraceHandler` wrapper class that stores a wrapped object. Its `invoke` method simply prints the name and parameters of the method to be called and then calls the method with the wrapped object as the implicit parameter.

```java
class TraceHandler implements InvocationHandler {
    private Object target;

    public TraceHandler(Object t) {
        target = t;
    }
}
```
public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
    // print method name and parameters
    // invoke actual method
    return m.invoke(target, args);
}

Here is how you construct a proxy object that causes the tracing behavior whenever one of its methods is called:

Object value = . . .;
// construct wrapper
InvocationHandler handler = new TraceHandler(value);
// construct proxy for one or more interfaces
Class[] interfaces = new Class[] { Comparable.class};
Object proxy = Proxy.newProxyInstance(null, interfaces, handler);

Now, whenever a method from one of the interfaces is called on proxy, the method name and parameters are printed out and the method is then invoked on value.

In the program shown in Listing 6.9, we use proxy objects to trace a binary search. We fill an array with proxies to the integers 1 . . . 1000. Then we invoke the binarySearch method of the Arrays class to search for a random integer in the array. Finally, we print the matching element.

Object[] elements = new Object[1000];
// fill elements with proxies for the integers 1 . . . 1000
for (int i = 0; i < elements.length; i++) {
    Integer value = i + 1;
    elements[i] = Proxy.newProxyInstance(. . .); // proxy for value;
}

// construct a random integer
Integer key = new Random().nextInt(elements.length) + 1;

// search for the key
int result = Arrays.binarySearch(elements, key);

// print match if found
if (result >= 0) System.out.println(elements[result]);

The Integer class implements the Comparable interface. The proxy objects belong to a class that is defined at runtime. (It has a name such as $Proxy0.) That class also implements the Comparable interface. However, its compareTo method calls the invoke method of the proxy object’s handler.
NOTE: As you saw earlier in this chapter, the Integer class actually implements Comparable<Integer>. However, at runtime, all generic types are erased and the proxy is constructed with the class object for the raw Comparable class.

The binarySearch method makes calls like this:

```java
if (elements[i].compareTo(key) < 0) . . .
```

Since we filled the array with proxy objects, the compareTo calls call the invoke method of the TraceHandler class. That method prints the method name and parameters and then invokes compareTo on the wrapped Integer object.

Finally, at the end of the sample program, we call

```java
System.out.println(elements[result]);
```

The println method calls toString on the proxy object, and that call is also redirected to the invocation handler.

Here is the complete trace of a program run:

```
500.compareTo(288)
250.compareTo(288)
375.compareTo(288)
312.compareTo(288)
281.compareTo(288)
296.compareTo(288)
288.compareTo(288)
288.toString()
```

You can see how the binary search algorithm homes in on the key by cutting the search interval in half in every step. Note that the toString method is proxied even though it does not belong to the Comparable interface—as you will see in the next section, certain Object methods are always proxied.

Listing 6.9  proxy/ProxyTest.java

```java
package proxy;

import java.lang.reflect.*;
import java.util.*;

/**
 * This program demonstrates the use of proxies.
 * @version 1.00 2000-04-13
 * @author Cay Horstmann
 */
(Continues)
```
public class ProxyTest
{
    public static void main(String[] args)
    {
        Object[] elements = new Object[1000];

        // fill elements with proxies for the integers 1 ... 1000
        for (int i = 0; i < elements.length; i++)
        {
            Integer value = i + 1;
            InvocationHandler handler = new TraceHandler(value);
            Object proxy = Proxy.newProxyInstance(null, new Class[] { Comparable.class }, handler);
            elements[i] = proxy;
        }

        // construct a random integer
        Integer key = new Random().nextInt(elements.length) + 1;

        // search for the key
        int result = Arrays.binarySearch(elements, key);

        // print match if found
        if (result >= 0) System.out.println(elements[result]);
    }
}

/**
 * An invocation handler that prints out the method name and parameters, then
 * invokes the original method
 */
class TraceHandler implements InvocationHandler
{
    private Object target;

    /**
     * Constructs a TraceHandler
     * @param t the implicit parameter of the method call
     */
    public TraceHandler(Object t)
    {
        target = t;
    }

    public Object invoke(Object proxy, Method m, Object[] args) throws Throwable
    {
        // print implicit argument
        System.out.print(target);
    }
6.5.1 Properties of Proxy Classes

Now that you have seen proxy classes in action, let’s go over some of their properties. Remember that proxy classes are created on the fly in a running program. However, once they are created, they are regular classes, just like any other classes in the virtual machine.

All proxy classes extend the class `Proxy`. A proxy class has only one instance field—the invocation handler, which is defined in the `Proxy` superclass. Any additional data required to carry out the proxy objects’ tasks must be stored in the invocation handler. For example, when we proxied `Comparable` objects in the program shown in Listing 6.9, the `TraceHandler` wrapped the actual objects.

All proxy classes override the `toString`, `equals`, and `hashCode` methods of the `Object` class. Like all proxy methods, these methods simply call `invoke` on the invocation handler. The other methods of the `Object` class (such as `clone` and `getClass`) are not redefined.

The names of proxy classes are not defined. The `Proxy` class in Oracle’s virtual machine generates class names that begin with the string `$Proxy`.

There is only one proxy class for a particular class loader and ordered set of interfaces. That is, if you call the `newProxyInstance` method twice with the same class loader and interface array, you get two objects of the same class. You can also obtain that class with the `getProxyClass` method:

```java
Class proxyClass = Proxy.getProxyClass(null, interfaces);
```

A proxy class is always `public` and `final`. If all interfaces that the proxy class implements are `public`, the proxy class does not belong to any particular package.
Otherwise, all non-public interfaces must belong to the same package, and the proxy class will also belong to that package.

You can test whether a particular Class object represents a proxy class by calling the isProxyClass method of the Proxy class.

```java
java.lang.reflect.InvocationHandler 1.3

- Object invoke(Object proxy, Method method, Object[] args)
  define this method to contain the action that you want carried out whenever a method was invoked on the proxy object.
```

```java
java.lang.reflect.Proxy 1.3

- static Class getProxyClass(ClassLoader loader, Class[] interfaces)
  returns the proxy class that implements the given interfaces.

- static Object newProxyInstance(ClassLoader loader, Class[] interfaces, InvocationHandler handler)
  constructs a new instance of the proxy class that implements the given interfaces. All methods call the invoke method of the given handler object.

- static boolean isProxyClass(Class c)
  returns true if c is a proxy class.
```

This ends our final chapter on the fundamentals of the Java programming language. Interfaces and inner classes are concepts that you will encounter frequently. However, as we already mentioned, proxies are an advanced technique that is of interest mainly to tool builders, not application programmers. You are now ready to go on to learn about graphics and user interfaces, starting with Chapter 7.
Symbols
- (minus sign)
  arithmetic operator, 55, 63
  printf flag, 81
-- operator, 56, 63
_ (underscore)
  delimiter, in number literals, 47
  in instance field names (C++), 175
, (comma)
  operator (C++), 63
  printf flag, 81
; (semicolon)
  for statements, 45, 52
  in class path (Windows), 190
: (colon)
  in assertions, 653
  in class path (UNIX), 190
inheritance token (C++), 204
:: operator (C++), 153, 160, 207
! operator, 57, 63
!= operator, 57, 63
  for floating-point numbers, 97
?: operator, 58, 63
/ (slash)
  arithmetic operator, 55, 63
  in file names, 571
// comments, 46
/*...*/ comments, 46
/*/...*/ comments, 46, 193–194
. (period)
  in class path, 190–191
  in directory names (UNIX), 611
... (ellipsis), in varargs, 255
^ operator, 58, 63
~ operator, 58, 63
' (single quote escape sequence), 50
" (double quote escape sequence), 50
"..." (for strings), 45
( (left parenthesis), printf flag, 81
) (empty parentheses), in method calls, 45
(...) (parentheses)
  for casts, 61, 63
  for operator hierarchy, 62–63
[...] (square brackets), for arrays, 107, 112
{...} (curly braces)
  double, in inner classes, 321
  for blocks, 44–45, 86
  for enumerated type, 63–64
@ (at), in javadoc comments, 194–195
$ (dollar sign)
  delimiter, for inner classes, 312
  in variable names, 52
  printf flag, 81
* (asterisk)
  arithmetic operator, 55, 63
  echo character, 447
  in class path, 191
  in imports, 183
\ (backslash)
  escape sequence for, 50
  in file names, 84, 571
$ (dollar sign)
  delimiter, for inner classes, 312
  in variable names, 52
  printf flag, 81
% (percent sign)
  arithmetic operator, 55, 63
  for casts, 61, 63
  for different numeric types, 61
  for concatenating
    objects and strings, 237–238
    strings, 64–65
    printf flag, 81
+ (plus sign)
  arithmetic operator, 55, 63
  for concatenating
    objects and strings, 237–238
    strings, 64–65
    printf flag, 81
++ operator, 56, 63
+ (plus sign)
  arithmetic operator, 55, 63
  for concatenating
    objects and strings, 237–238
    strings, 64–65
    printf flag, 81
& (ampersand)
  bitwise operator, 58, 63
  in bounding types, 705
  in reference parameters (C++), 171
&& operator, 57, 63
# (number sign)
  in javadoc hyperlinks, 197
  in property files, 395
  printf flag, 81
% (percent sign)
  arithmetic operator, 55, 63
  formatting output for, 80
  printf flag, 81
+ (plus sign)
  arithmetic operator, 55, 63
  for different numeric types, 61
  for concatenating
    objects and strings, 237–238
    strings, 64–65
    printf flag, 81
++ operator, 56, 63
< (left angle bracket)
  in shell syntax, 85
  printf flag, 81
? (in wildcard types), 723
<<, >>, >>> operators, 58, 63
<= operator, 57, 63
<..<> (angle brackets), for type parameters, 243, 700
> (right angle bracket)
  in shell syntax, 85, 679
  relational operator, 57, 63
>& (in shell syntax), 679
>= operator, 57, 63
= operator, 53, 55
== operator, 57, 63
  for class objects, 260
  for enumerated types, 256
  for floating-point numbers, 97
  for identity hash maps, 786
  for strings, 67
  for wrappers, 252
| operator, 58, 63
|| operator, 57, 63
0 (zero)
  prefix (octal integers), 47
  printf flag, 81
0b prefix (binary integers), 47
0x prefix (hexadecimal integers), 47
2> (in shell syntax), 679
2D shapes, 356–365

A
Absolute positioning, 516
Abstract classes, 221–226, 788
  and interfaces, 294–295
  extending, 222
  object variables of, 223
abstract keyword, 221–222
Abstract methods, 222
AbstractAction class, 405, 408, 474, 477
AbstractButton class, 422, 476–478
  isSelected, setSelected methods, 478
setAction method, 476
setActionCommand method, 458
setDisplayMnemonicIndex method, 481, 483
setHorizontalTextPosition method, 476–477
setMnemonic method, 483
abstractClasses/Employee.java, 225
abstractClasses/Person.java, 225
abstractClasses/PersonTest.java, 224
abstractClasses/Student.java, 226
AbstractCollection class, 749, 758, 790
AbstractList class, 790
AbstractMap class, 790
AbstractQueue class, 745, 790
AbstractSequentialList class, 790
AbstractSet class, 790
ACCELERATOR_KEY action table name, 404
Accelerators, 481–482
accept method (FileFilter), 548, 557
Access modifiers
  checking, 263
  final, 54, 157, 217–218, 292, 315–318, 867–868
  private, 150, 157–164, 227, 309
  protected, 227, 282, 299
  public static final, 293
  static, 44, 157–164
  static final, 54
void, 44
Access order, 784
AccessibleObject class
  isAccessible method, 274
  setAccessible method, 270, 274
Accessor methods, 153–154, 725
Accessory components, 550
Action interface, 403–411, 474
  actionPerformed method, 404
  addPropertyChangeListener, removePropertyChangeListener methods, 404–405
getValue, putValue methods, 404, 410
isEnabled, setEnabled methods, 404, 410
Action listeners, 403–411
addActionFrame, 409
ACTION_COMMAND_KEY action table name, 404
ActionEvent class, 384, 421
  getActionCommand method, 393, 422
getAttribute method, 422
ActionListener interface, 422
overriding, 474
implementing, 385, 393
ActionMap class, 407–408
Actions, 383, 403–411
  associating with keystrokes, 406
  predefined names for, 404
ActiveX, 6, 16, 600
Adapter classes, 399–403
add method
  of ArrayList, 244–247, 249
  of BigDecimal/BigInteger, 106
  of BlockingQueue, 878–879
  of ButtonGroup, 458
of Collection, 745, 750, 788
of Container, 387, 389, 436
of GregorianCalendar, 138, 144
of HashSet, 765
of JFrame, 350, 355
of JMenu, 474–475
of JToolBar, 489–491
of List, 761, 789
of ListIterator, 755–756, 761, 790
of Queue, 775
of Set, 790
addAll method
of ArrayList, 699
of Collection, 749–750
of Collections, 807
of TreeSet, 770
addChoosableFileFilter method (JFileChooser), 556
addComponent, addGroup methods ( GroupLayout), 515
addFirst, addLast methods
of Deque, 775
of LinkedList, 762
addHandler method (Logger), 674
addItem method ( JComboBox), 463–466
Addition operator, 55
addLayoutComponent method ( LayoutManager), 520
addPropertyChangeListener method (Action), 404–405
addSeparator method
of JMenu, 474–475
of JToolBar, 489–491
addShutdownHook method (Runtime), 181
addSuppressed method ( Throwable), 645, 648
AdjustmentEvent class, 421
getAdjustable, getAdjustmentType, getValue
methods, 422
AdjustmentListener interface, 422
adjustmentValueChanged method, 422
after method (Date), 137
Aggregation, 129–131
Algorithms, 126
for collections, 801–809
binary search, 805–806
simple, 806–808
QuickSort, 113, 803
writing, 808–809
Algorithms + Data Structures = Programs
(Wirth), 126
Algorithms in C++ (Sedgewick), 803
Alice in Wonderland (Carroll), 765, 768
addAll method (EnumSet), 787
Alt+F4, 482
and, andNot methods ( BitSet), 814
Andreessen, Mark, 11
Annotations, 711–712
Anonymous arrays, 110
Anonymous inner classes, 305, 318–322
simplifying code with, 391–392
anonymousInnerClass/AnonymousInnerClassTest.java, 320
Antisymmetry rule, 292
append method
of JTextArea, 451, 923
of StringBuilder, 74–75
appendCodePoint method (StringBuilder), 75
Applet class, 592
destroy method, 596
g.getAppletContext method, 608–610
g.getAppletInfo method, 606
getCodeBase, getDocumentBase methods, 606–607
gImage, getAudioClip methods, 607
getParameter method, 600–601, 606
getParameterInfo method, 606
init method, 595–596, 600
play method, 607
resize method, 596
showStatus method, 609–610
start, stop methods, 596
applet element ( HTML), 593, 596–599
align attribute, 597
alt attribute, 599
archive attribute, 598
code attribute, 597–598
codebase attribute, 597–598
name attribute, 599
object attribute, 598
vspace, hspace attributes, 597
width, height attributes, 595, 597
AppletContext interface, 607–610
gGetApplet, getApplets methods, 608, 610
showDocument method, 609–610
applet/NotHelloWorld.java, 593
Applets, 9–10, 591–610
aligning, 597
building, 36
changing warning string in, 189
communication between, 599, 608
context of, 607–610
converting applications to, 595–596
debugging, 594
displaying in browser, 591, 609–610
image and audio files in, 606–607
Applets (cont.)
  multiple copies of, 603
  no title bars for, 595
  passing information to, 599–606
  printing, 585
  running, 36, 591, 593
    in browser, 9, 13, 16, 593
    in IDE, 38, 594
  serialized objects of, 598
  sizing, 597–599
  testing, 593–594
  appletviewer program, 36, 593–594
Application Programming Interfaces (APIs),
  online documentation, 70, 72
Applications
  cache of, 577
  compiling for debugging, 691
  converting to applets, 595–596
  extensible, 216
  internationalization of, 571, 662
  launching, 43
  monitoring and management in JVM, 680
  preferences of, 610–624
  terminating, 44
  testing, 652–657
  troubleshooting, 27–28
Arguments. See Parameters
Arithmetic operators, 55–64
  autoboxing with, 252
  binary, 55
  precedence of, 63
Array class, 274–278
  getLength method, 275–276
  newInstance method, 275–276
Array lists, 108, 762–763
  anonymous, 321
  capacity of, 244
  elements of
    accessing, 246
    adding, 244–247
    removing, 247
    traversing, 248
  generic, 242–251
  raw vs. typed, 249–251
  size of, 244
    adjusting, 245
Array variables, 107, 318
ArrayBlockingQueue class, 879, 883
ArrayDeque class, 744, 774, 776
  as concrete collection type, 752, 791
ArrayIndexOutOfBoundsException, 108, 629–631, 909
ArrayList class, 109, 242–251, 698–700, 752, 789
  add method, 244–247, 249
  addAll method, 699
  as concrete collection type, 752, 791
  ensureCapacity method, 244, 246
  get, set methods, 246, 249
  remove method, 247, 249
  size method, 245–246
  synchronized, 889
  toArray method, 716
  trimToSize method, 245–246
arrayList/ArrayListTest.java, 248
Arrays, 107–123
  and type parameters, 713
  anonymous, 110
  array entries, numbering of, 108
  circular, 743, 745
  cloning, 302
  converting to collections, 800–801
  copying, 111–112
    on write, 888
  creating, 108
  element types of, 214
  elements of
    numbering, 108
    removing from the middle, 752–753
    traversing, 108–109, 118
  equality testing for, 233
  generic, 274–278, 722
  hash codes of, 236
  in command-line parameters, 112–113
  initializing, 108, 110
  multidimensional, 116–122, 238
  of integers, 238
  of wildcard types, 713
  out-of-bounds access in, 629
  printing, 118, 238
  ragged, 120–123
  size of, 109, 275
    equal to 0, 111, 800
    equal to 1, 318
    increasing, 111
    setting at runtime, 243
  sorting, 113–116, 289
Arrays class
  asList method, 793, 798, 800
  binarySearch method, 116, 328
  copyOf method, 111, 115, 274
  copyOfRange method, 115
  deepToString method, 118, 238
  equals method, 116, 233
fill method, 116
hashCode method, 236
sort method, 113, 116, 286, 289, 291
toString method, 110, 115
arrays/CopyOfTest.java, 277
ArrayStoreException, 713–714, 722
Ascender, Ascent (in typesetting), 372
ASCII standard, 50, 371
asList method (Arrays), 793, 798, 800
assert keyword, 652–657
Assertions, 652–657
defining parameters with, 655–656
documented assumptions with, 656–657
enabling/disabling, 653–654
Assignment operator, 55
Associations, 887
Asynchronous methods, 890
Atomicity, 868
AtomicPrimitiveType interfaces, 868
AtomicReference interface, 868
Audio files, accessing from applets, 606–607
@author comment (javadoc), 196, 198
AutoBoxing, 251–254
AutoCloseable interface, 644
close method, 644–645
await method (Condition), 838, 855–859, 873–874
awaitUninterruptibly method (Condition), 873–874
AWT (Abstract Window Toolkit), 334
events in
  debugging, 686–690
  hierarchy of, 419–423
  tracing, 683
  preferred field sizes in, 444
AWTEvent class, 420

B
\b (backspace escape sequence), 50
Background
default color for, 366–367
  erasing, 822
  painting, 354
BadCastException, 731
Barriers, 907–908
Base classes. See Superclasses
Baseline (in typesetting), 372, 510
bash shell, setting CLASSPATH in, 28
Basic multilingual planes, 50
Basic, procedure-oriented language, 4
BasicButtonUI class, 433
BasicService interface, 584
getCodeBase method, 585, 589
isWebBrowserSupported method, 589
showDocument method, 589
Batch files, 192
Beans, 566
before method (Date), 137
BIG-5 standard, 50
BigDecimal/BigInteger classes, 105–107
  add, subtract, multiply, divide, mod methods, 106
compareTo method, 106
valueOf method, 105–106
BigIntegerTest/BigIntegerTest.java, 107
Binary search, 805–806
binarySearch method
  of Arrays, 116, 328
  of Collections, 805–806
Bit masks, 58, 412
Bit sets, 813–817
Bitcode files, 43
BitSet interface, 741, 810, 813–817
  and, or, xor, andNot methods, 814
  get, set, clear, length methods, 813
  “Sieve of Eratosthenes” benchmark, 814–817
Bitwise operators, 58
precedence of, 63
Black box behavior, of objects, 128
Blank final variables, 317
Blank lines, printing, 45
Blocking queues, 877–885
BlockingDeque interface
  offerFirst, offerLast methods, 885
  pollFirst, pollLast methods, 885
  putFirst, putLast methods, 885
  takeFirst, takeLast methods, 885
BlockingQueue interface
  add, element, peek, remove methods, 878–879
  offer, poll, put, take methods, 878–879, 885
blockingQueue/BlockingQueueTest.java, 880
Blocks, 44, 86–87
  nested, 86
Boolean class, 251
boolean operators, 57
  precedence of, 63
boolean type, 51
  default initialization of, 172
  formatting output for, 80
  no casting to numeric types for, 62
BooleanHolder class, 253
Border layout manager, 437–439
border/BorderFrame.java, 459
BorderFactory class, 458–462
  createTypeBorder methods, 458–462
BorderLayout class, 437–439
  constants of, 437
Borders, 458–462
bounce/BallComponent.java, 826
bounce/Ball.java, 824
bounce/Bounce.java, 822
bounceThread/BounceThread.java, 830
Bounded collections, 745
Bounding rectangle, 359–360
Bounds checking, 112
Bourne Again shell, setting paths in, 20, 28
Box layout, 493
break statement, 101–105
  labeled, 103
  missing, 680
  unlabeled, 102
Bridge methods, 709–711, 721
brighter method (Color), 367
BrokenBarrierException, 908
Browsers
  and MIME types, 574
  configuring for Java Web Start, 576
  default, 584
  installing Java Plug-in in, 591
Java-enabled, 565, 599
  persistent, 598
  running applets in, 9, 13, 16, 593–594, 607
  status bar of, 609–610
  web page display area of, 609–610
Buckets, 764
Bulk operations, 799–800
button/ButtonFrame.java, 390
ButtonGroup class, 454
  add method, 458
  getSelection method, 456, 458
ButtonModel interface, 432–433
  getActionCommand method, 456, 458
  getSelectedObjects method, 456
  properties of, 433
Buttons
  associating actions with, 405
  creating, 387
  handling clicks for, 386–391
  model-view-controller analysis of, 432–433
  rearranging automatically, 435
ButtonUIListener class, 433
Byte class, 251
byte type, 47
ByteArrayOutputStream class, 583

C

C
  assert macro in, 654
  event-driven programming in, 384
  procedure-oriented, 4
C#, 9, 15
  delegates in, 278
  polymorphism in, 218
  properties in, 345
  useful features of, 12
C++
  , operator in, 63
  :: operator in, 153, 207
  = 0 in, 223
  >> operator in, 59
  accessor methods in, 139
  algorithms in, 802
  applicability of, 14
  arrays in, 112, 123
  bitset template in, 813
  boolean values in, 51
  class-based access privileges in, 156
  classes in, 44
    nested, 306
  code units and code points in, 69
  control flow in, 86
  copy constructors in, 136
  destructor methods in, 181
  dynamic casts in, 220
  exceptions in, 629, 632–633, 637
  for loop in, 96
  function pointers in, 278, 303
  #include in, 184
  inheritance in, 204, 294–295
  instance fields in, 174–175
  integer types in, 47
  namespace directive in, 184
  new operator in, 151
  NULL pointer in, 135
  object pointers in, 135
  operator overloading in, 105
  passing parameters in, 167, 171
  performance of, comparing to Java, 817
  polymorphism in, 218
  protected modifier in, 227
  pure virtual functions in, 223
  redefining variables in nested blocks in, 87
  references in, 135
  Standard Template Library in, 742, 747
  static fields and methods in, 160
  static member functions in, 44
strings in, 66–67
superclasses in, 208
syntax of, 3
templates in, 12, 701, 704, 707
this pointer in, 176
type parameters in, 704
using directive in, 184
using iterators as parameters in, 811
variables in, 54
vector template in, 245
virtual constructors in, 261
void* pointer in, 228

C shell, setting CLASSPATH in, 28
Cache, 577
calculator/CalculatorPanel.java, 440
Calendar class, 136
constants, 137–138
getime, setTime methods, 217
Calendars
displaying, 139–141
vs. time measurement, 137
weekday names in, 140
CalendarTest/CalendarTest.java, 142
Call by reference, 164
Call by value, 164–171
Callable interface, 901
call method, 890, 894
wrapper for, 890
Callables, 889–895
Callbacks, 302–305
Camel case (CamelCase), 43
cancel method (Future), 890, 894, 896, 916
CancellationException, 916
Canned functionality, 905
canRead, canWrite methods (FileContents), 590
Carriage return, escape sequence, 50
case statement, 101
cast method (Class), 731–734
Casts, 61–62, 218–220
bad, 629
checking before attempting, 219
catch statement, 634–649
ceiling method (NavigableSet), 774
ChangeListener interface, 466
stateChanged method, 466–467
char type, 49–51
Character class, 251
isJavaIdentifierStart, isJavaIdentifierPart methods, 52
Characters, formatting output for, 80
charAt method (String), 69–70
chart/Chart.java, 603
checkBox/CheckBoxTest.java, 453
Checkboxes, 452–454
in menus, 477–478
Checked exceptions, 260–262, 629–632
applicability of, 651
declaring, 630–632
defeating with generics, 718–720
Checked views, 796
checkedCollection methods (Collections), 798
Child classes. See Subclasses
Choice components
borders, 458–462
checkboxes, 452–454
combo boxes, 462–466
radio buttons, 454–458
sliders, 466–473
circleLayout/CircleLayoutFrame.java, 520
circleLayout/CircleLayout.java, 517
Circular arrays, 743, 745
Clark, Jim, 11
Clarke, Arthur C., 508
Class class, 259, 731–734
cast method, 731–732
forName method, 259, 263
generic, 715
getClass method, 259
getConstructor, getDeclaredConstructor methods, 732
getConstructors method, 263, 268
getDeclaredConstructors method, 264, 268
getDeclaredField method, 274
getDeclaredFields method, 264, 268, 270, 274
getDeclaredMethods method, 264, 268, 279
getEnumConstants method, 731–732
getField method, 274
getFields method, 263, 268, 274
getGenericSuperclass, getGenericInterfaces methods, 737
getImage, getAudioClip methods, 570
getMethod method, 279
getMethods method, 263, 268
getName method, 242, 259–260
getResource, getResourceAsStream methods, 570–571, 573
getSuperclass method, 242, 732
getTypeParameters method, 737
newInstance method, 261, 263, 731–732
Class constants, 54
Class diagrams, 130–131
Class fields. See Static fields
.class file extension, 43, 596
Class files, 185, 190
   locating, 191–192
   names of, 43, 148
class keyword, 42
Class loaders, 327, 653–654
Class path, 190–193
   checking directories on, 680
   setting, 192–193
ClassCastException, 219, 275, 292, 712, 722, 796
Classes, 127–128, 204–227
   abstract, 221–226, 294–295, 788
   access privileges for, 156
   adapter, 399–403
   adding to packages, 185–188
   analyzing
      capabilities of, 263–269
      objects of, at runtime, 269–274
   concrete, 790–791
   constructors for, 149
   defining, 145–157
      at runtime, 326
   deprecated, 196
   design hints for, 199–201
   digitally signed, 6, 579–582
documentation comments for, 193–196
   encapsulation of, 127, 153–155
   extending, 128
   final, 217–218
   generic, 243–244, 463, 700–702
      extending/implementing other generic
classes, 723
grouping. See Packages
helper, 499–505
   identifying, 129
   immutable, 157
   implementing multiple interfaces, 293
   importing, 182–184
inner, 305–326, 391–393
   instances of, 127, 132
   loading, 260
   multiple source files for, 148–149
   names of, 27, 42, 182, 201
      full package, 182
   number of basic types in, 199
   package scope of, 188
   parameters in, 152–153
   predefined, 131–145
   private methods in, 156–157
   protected, 227
public, 193
relationships between, 129–131
   runtime type identification of, 259
   serializable, 680
   sharing, among programs, 190
   subclasses, 204, 222–227, 321
   superclasses, 204, 222–227, 282, 292, 299, 632, 637
unit testing, 162
ClassLoader class, 657
   clearAssertionStatus method, 657
   setDefaultCloseOperationStatus method, 657
   setClassAssertionStatus method, 657
   setPackageAssertionStatus method, 657
CLASSPATH environment variable, 28, 192–193
Class<T> parameters, 732
clear method
   of BitSet, 813
   of Collection, 749–750
Client-side locking, 864
close method
   of AutoCloseable, 644–645
   of Closeable, 645
Closeable interface, 645
Code errors, 627
Code points, 51
Code units, 51, 68
Codebase, 585
codePointAt method (String), 69–70
codePointCount method (String), 68, 71
Collection interface, 745, 748–750, 758, 788
   add method, 745, 749–750, 788
   addAll method, 749–750
   clear method, 749–750
   contains, containsAll methods, 749–750, 758
   equals method, 749
generic, 748
isEmpty method, 749–750
iterator method, 745, 750
remove, removeAll methods, 749–750
retain method, 749–750
size method, 749–750
toArray method, 247, 749–750
Collections, 741–817
  algorithms for, 801–809
  bounded, 745
  concrete, 751–787
  concurrent modifications of, 757
  converting to arrays, 800–801
  debugging, 758
  elements of
    inserting, 787
    maximum, 801
    removing, 748
    traversing, 746
  interfaces for, 741–751, 787–801
  legacy, 791, 809–817
  ordered, 754, 789
  searching in, 805–806
  sorted, 767
  thread-safe, 796, 885–889
  type parameters for, 700
  using for method parameters, 809
Collections class, 793, 804
  addAll method, 807
  binarySearch method, 805–806
  checkedCollection methods, 798
  copy method, 807
  disjoint method, 808
  fill method, 807
  frequency method, 808
  indexOfSubList, lastIndexOfSubList methods, 807
  min, max methods, 807
  nCopies method, 793, 798
  replaceAll method, 807
  reverse method, 807
  reverseOrder methods, 803, 805
  rotate method, 808
  shuffle methods, 804–805
  singleton, singletonList, singletonMap methods, 793, 798
  sort methods, 802–805
  swap method, 807
  synchronizedCollection methods, 796, 798, 889
  unmodifiableCollection methods, 795, 797
Collections framework, 787–801
  algorithms, 801–809
  classes, 790
  converting to arrays, 800–801
  legacy collections, 810
  operations
    bulk, 799–800
    optional, 797
  views and wrappers, 792
Color choosers, 557–563
Color class, 365
  brighter, darker methods, 367
  predefined constants, 366
ColorAction class, 404
colorChooser/ColorChooserPanel.java, 561
Colors
  background, 354, 366–367
  changing, 405
  custom, 365
  foreground, 366
  predefined, 367–368
  system, 368
Columns (of text field), 444
Combo boxes, 462–466
  adding items to, 463–464
  current selection in, 463
comboBox/ComboBoxFrame.java, 465
Command-line parameters, 112–113
Command-line tools, 25
Comments, 45–46
  blocks of, 46
  for automatic documentation, 46, 193–199
  in property files, 395
  not nesting, 46
  to the end of line, 46
Comparable interface, 286, 328, 704–705, 768, 773–774, 803
comparator method (SortedMap), 773, 782
Comparator interface, 769, 773, 803
  equals method, 770
compare method
  of Comparable, 768
  of Comparator, 769–770, 773
  of Double/Integer, 291
compareTo method
  in subclasses, 292
  of BigDecimal/BigInteger, 106
  of Comparable, 286, 288, 290, 704, 727, 768, 773
  of Employee, 287
  of Enum, 257
  of String, 70
Compilation errors, 32
Compiler
  autoboxing in, 252
  bridge methods in, 709
Compiler (cont.)
command-line options of, 680
creating bytecode files in, 43
deducing method types in, 703
enforcing throws specifiers in, 636
error messages in, 32, 631
just-in-time, 6–7, 15, 153, 682, 817
launching, 26
optimizing method calls in, 8, 218
shared strings in, 65, 67
translating inner classes in, 312
type parameters in, 699
whitespace in, 44
Component class, 422
gSetFont method, 446
getPreferredSize method, 353, 355
getSize method, 348
inheritance hierarchies of, 435
isVisible method, 348
repaint method, 352, 355
setBackground, setForeground methods, 366, 369
setBounds method, 342, 344, 348, 516
setCursor method, 419
setLocation method, 342, 344, 348
setSize method, 348
setVisible method, 341–342, 348, 923
validate method, 446, 923
Components, 434
displaying information in, 349
labeling, 446–447
realized, 923
Composite design pattern, 427
Compound statements. See Blocks
CompoundInterest/CompoundInterest.java, 119
com.sun.java package, 395
Concrete classes, 790–791
Concrete collections, 751–787
Concrete methods, 222
Constructors, 149–151, 171–181
calling another constructor in, 176
default field initialization in, 172–173
defined, 132
documentation comments for, 193
explicit field initialization in, 174
final, 263
initialization blocks in, 176–181
names of, 133, 150
no-argument, 173, 207
overloading, 171–172
parameter names in, 175
private, 263
protected, 193
public, 193, 263
with super keyword, 207
ConstructorTest/ConstructorTest.java, 179
Consumer threads, 878
Console class
reading passwords with, 77
readLine, readPassword methods, 78
console method (System), 78
ConsoleHandler class, 663–667, 675
ConsoleWindow class, 682
const keyword, 55
Constants, 54–55
documentation comments for, 196
names of, 54
public, 55, 159
static, 158–159
Constructor class, 263
gGetDeclaringClass method, 268
gExceptionTypes method, 268
gModifiers method, 263, 268
gGetName method, 263, 268
gGetParameterTypes method, 268
newInstance method, 263, 732
signal, signalAll methods, 869
vs. synchronization methods, 861
Condition objects, 854–859
Condition variables, 854
Conditional statements, 87–90
cfg method (Logger), 659, 673
Configuration files, 616–624
Consumer threads, 878
Container class, 435
add method, 389, 436
setLayout method, 436
Containers, 434
contains method
  of Collection, 749–750, 758
  of HashSet, 765
containsAll method (Collection), 749–750, 758
containsKey, containsValue methods (Map), 781
Content pane, 350
continue statement, 103–105
Control flow
  block scope, 86–87
  breaking, 102–105
  conditional statements, 87–90
  loops, 91–95
    determinate, 95–99
    “for each”, 109–110
  multiple selections, 99–102
Controllers, 429
Conversion characters, 80
Cooperative scheduling, 837
Coordinated Universal Time (UTC), 136
copy method (Collections), 807
copyArea method (Graphics), 378, 381
copyOf method (Arrays), 111, 115, 274
copyOfRange method (Arrays), 115
CopyOnWriteArrayList class, 888–889
CopyOnWriteArraySet class, 888
Core Java program examples, 23
Corruption of data, 843–849
Count of Monte Cristo, The (Dumas), 768, 915–917
Countdown latches, 907
CountDownLatch class, 906–907
Covariant return types, 710
create method
  of EventHandler, 394–395
  of PersistenceService, 585, 590
createCustomCursor method (Toolkit), 414, 419
createDialog method (JColorChooser), 563
createFont method (Font), 371
createParallelGroup method (GridLayout), 514
createScreenCapture method (Robot), 690
createTypeBorder methods, 458–462
Ctrl+\, 869
current method (ThreadLocalRandom), 872
Current user, 617
currentThread method (Thread), 833–836
Cursor class, 413
  getPredefinedCursor method, 413
Cursor shapes, 414
Custom layout managers, 516–520
CyclicBarrier class, 906–908

D
D suffix (double numbers), 48
Daemon threads, 841
darker method (Color), 367
Data exchange, 538–545
Data fields
  initializing, 176–545
  public, 150
Data types, 46–51
  boolean type, 51
  casting between, 62
  char type, 49–51
  conversions between, 218–220
  floating-point, 48–49, 60–61
  integer, 47–48, 60–61
Databases, closing connections to, 640
dataExchange/DataExchangeFrame.java, 541
dataExchange/PasswordChooser.java, 542
Date and time
  formatting output for, 80–81
  no built-in types for, 132
Date class, 136
  and GregorianCalendar class, 139
  before, after methods, 137
  getDay, getMonth, getYear methods, 137
  toString method, 133
DateFormatSymbols class, 140, 145
  getShortWeekdays, getShortMonths, getWeekdays,
    getMonths methods, 141, 145
DateInterval class, 709
Deadlocks, 856, 868–871, 875
  breaking up, 873
debugger/BuggyButtonTest.java, 691
Debugging, 8, 677–682
  AWT events, 683, 686–690
  debuggers for, 677, 690–695
  generic types, 796
  GUI programs, 635
  including class names in, 322
  intermittent bugs, 67, 341, 924
  messages for, 634
  of collections, 758
  reflection for, 270
  trapping program errors in a file for, 679
DebugGraphics class, 683
Decorator design pattern, 427
Decrement operators, 56–57
Deep copies, 296, 298
deepToString method (Arrays), 118, 238
DEFAULT action table name, 404
default statement, 101

Index
DefaultButtonModel class, 432
DefaultComboBoxModel class, 464
delay method (Robot), 690
Delayed interface, 879
getDelay method, 880, 884
DelayQueue class, 879–880, 884
Delegates, 278
delete method
  of PersistenceService, 590
  of StringBuilder, 75
Dependence, 129–131
@deprecated comment (javadoc), 196
Deprecation classes, 196
Deprecation methods, 137–138, 196, 680
Deprecation variables, 196
Deque interface, 774–776
  addFirst, addLast methods, 775
  getFirst, getLast methods, 775
  offerFirst, offerLast methods, 775
  peekFirst, peekLast methods, 775
  pollFirst, pollLast methods, 775
  removeFirst, removeLast methods, 775
Deques, 774–776
Derived classes. See Subclasses
deriveFont method (Font), 371, 376
Descender (in typesetting), 372
descendingIterator method (NavigableSet), 774
Descent (in typesetting), 372
Design patterns, 426–428
Design Patterns—Elements of Reusable Object-Oriented Software (Gamma et al.), 428
destroy method (Applet), 596
Determinate loops, 95–99
Development environments
  choosing, 24–25
  in shell window, 27
  integrated, 28–33
Device errors, 627
dialog/AboutDialog.java, 537
dialog/DialogFrame.java, 536
Dialogs, 522–563
  accepting/canceling, 539
  centering, 305
  closing, 399–403, 482, 535
  color choosers, 557–563
  creating, 533–538
  data exchange, 538–545
  default button in, 540
  displaying, 535
  file, 545–557
  maximized, 399
  modality of, 522, 534, 539
  option, 523–533
  pop-up, 579
  root pane of, 540
Diamond syntax, 243
Dijkstra, Edsger, 907
disjoint method (Collections), 808
divide method (BigDecimal / BigInteger), 106
Division operator, 55
Doclets, 199
docs directory, 24
Documentation comments, 46, 193–199
  extracting, 198–199
  for fields, 196
  for methods, 195
  for packages, 197
  general, 196
  hyperlinks in, 197
  inserting, 193–194
  links to other files in, 194
  overview, 198
  using HTML in, 194
doInBackground method (SwingWorker), 915, 917, 922
Do-nothing methods, 400
Double brace initialization, 321
Double buffering, 683
Double class, 49, 251
  compare method, 291
  POSITIVE_INFINITY, NEGATIVE_INFINITY, NaN constants, 49
double type, 48
  arithmetic computations with, 55
  converting to other numeric types, 60–61
Double-precision numbers, 48–49
Doubly linked lists, 752
do/while loop, 92, 95
draw method (Graphics2D), 357
draw/DrawTest.java, 362
drawImage method (Graphics), 378, 381
Drawing with mouse, 411–419
drawString method (Graphics / Graphics2D), 377
Drop-down lists, editable. See Combo boxes
Dynamic binding, 209, 214–217

E

  e (exponent), in numbers, 48
  E
    as type variable, 701
    constant (Math), 60
Echo character, 447–448
Eclipse, 25, 28–33, 677
configuring a project in, 30
debugging in, 690–695
editing a source file in, 30
error reports in, 32
imports in, 183
running applets in, 38, 594
SWT toolkit, 338
ECMA-262 (JavaScript subset), 15–16
Eiffel, multiple inheritance in, 294
element method
of BlockingQueue, 878–879
of Queue, 775
elements method
of Hashtable, 811
of Vector, 811
Ellington, Duke, 335
Ellipse2D class, 356, 360–361
setFrameFromCenter method, 361
setFrameFromDiagonal method, 360
Ellipse2D.Double class, 365
Ellipses, 360–361
else statement, 87, 89
else if statement, 89–90
Emacs text editor, 25
Employee class, 145–150, 204–209
compareTo method
main method, 162
no-argument constructor for, 173
toString method, 236–237
EmployeeTest/EmployeeTest.java, 146
EmptyStackException, 652
Encapsulation, 127
benefits of, 153–155
endsWith method (String), 70
ensureCapacity method (ArrayList), 244, 246
entering method (Logger), 673
Enterprise Edition (Java EE), 12, 19
entrySet method (Map), 778, 781
Enum class, 256–258
compareTo, ordinal, values methods, 257
toString, valueOf methods, 256–257
enum keyword, 63
Enumerated types, 63–64
equality testing for, 256
in switch statement, 102
Enumeration interface, 741, 810–811
nextElement, hasNextElements methods, 746, 810–811
Enumeration maps, 785
Enumeration sets, 785
Enumerations, 256–258, 608, 810–811
EnumMap class, 785, 787
as concrete collection type, 752
enums/EnumTest.java, 257
EnumSet class, 785, 787
allOf, noneOf, of, range methods, 787
as concrete collection type, 752
EOFException, 632
Epoch, 136
equals method
and hashCode method, 235
and inheritance, 230–233
for wrappers, 252
of Arrays, 116, 233
of Collection, 749
of Comparator, 770
of Object, 228–233, 242, 795
of proxy classes, 331
of Set, 790
of String, 67, 70
redefining, 235
equals/Employee.java, 240
equals/EqualsTest.java, 239
equalsIgnoreCase method (String), 67, 70
equals/Manager.java, 241
Error class, 628
Errors
checking, in mutator methods, 154
code, 627
compilation, 32
device, 627
internal, 628, 631, 655
messages for, 638
NoClassDefFoundError, 28
physical limitations, 627
ThreadDeath, 839, 843, 876
user input, 627
Escape sequences, 50
Etched borders, 458
Event delegation model, 384
Event dispatch thread, 341, 827, 876–877
and time-consuming tasks, 910
Event handling, 383–423
basics of, 383–403
semantic vs. low-level events, 421
summary of, 421–423
Event listeners, 384–385
with single method call, 394–395
Event objects, 384
Event procedures, 383
Event sources, 384–385
EventHandler class, 394
create method, 394–395
EventObject class, 384, 419
getActionCommand method, 421
getSource method, 393, 421
EventQueue class
invokeAndWait method, 911, 915
invokeLater method, 911, 915, 924
isDispatchThread method, 915
EventTracer class, 683, 685
eventTracer/EventTracer.java, 683
ExampleFileView class, 550
Exception class, 628–629, 648
Exception handlers, 261, 627
Exception specification, 630
Exceptions
ArrayIndexOutOfBoundsException, 108, 629–631, 909
ArrayStoreException, 713–714, 722
BadCastException, 731
CancellationException, 916
catching, 261–263, 631–632, 634–649
multiple, 637–638
changing type of, 639
checked, 260–262, 629–632, 651
ClassCastException, 219, 275, 292, 716, 722, 796
classification of, 628–629
CloneNotSupportedException, 298–299
ConcurrentModificationException, 755, 757, 886, 889
creating classes for, 633–634
documentation comments for, 195
EmptyStackException, 652
EOFException, 632
FileNotFoundException, 630, 632, 636, 643
finally clause in, 640–644
generics in, 718–720
hierarchy of, 628, 651
IllegalArgumentException, 270
IllegalStateException, 748, 751, 761, 775, 878
InterruptedException, 821, 828, 833–836, 873–874, 890, 894
IOException, 85, 630, 632, 636, 643
logging, 660, 669
micromanaging, 650
NoSuchElementException, 746, 751, 762, 775
NullPointerException, 629, 652
NumberFormatException, 651
propagating, 636, 652
rethrowing and chaining, 639, 678
RuntimeException, 628–629, 651
ServletException, 639
squelching, 651
stack trace for, 646–649
“throw early, catch late”, 652
throwing, 261–263, 632–633
InterruptedException, 890
tips for using, 649–652
UncaughtServiceException, 583
unchecked, 679, 839, 841–843
uncaught, 679, 839, 841–843
unchecked, 629–631, 651
unexpected, 660, 669
UnsupportedOperationException, 779, 793, 795, 797
variables for, implicitly final, 638
vs. simple tests, 649
wrapping, 639–640
Exchanger class, 906, 908
Exchangers, 908
.exe file extension, 569
Executable JAR files, 568–569
Executable path, 20–22
execute method (SwingWorker), 916, 922
ExecutorCompletionService class, 901
poll, submit, take methods, 902
Executors, 895–905
groups of tasks, controlling, 901–902
scheduled, 900–901
Executors class
newCachedThreadPool method, 895–896, 899
newFixedThreadPool method, 895–896, 900
newScheduledThreadPool method, 895, 900
newSingleThreadExecutor method, 895–896, 900
newSingleThreadScheduledExecutor method, 895, 900
ExecutorService interface, 896
invokeAny, invokeAll methods, 901–902
shutdown method, 896, 900
shutdownNow method, 896, 901
submit method, 896, 900
Exit code, 44
exit method (System), 44
exiting method (Logger), 659, 673
Explicit parameters, 152–153
exportSubtree, exportNode methods (Preferences), 618, 623
ExtendedService class, 584
extends keyword, 204–227, 704–705
External padding, 497
Index

F

F suffix (float numbers), 48
Factorial function, 646
Factory methods, 161
Fair locks, 854
Fallthrough behavior, 101
fdlibm (Freely Distributable Math Library), 60
Field class, 263
    getDeclaringClass method, 268
    getModifiers method, 263, 268
    getName method, 263, 268
    getType method, 263
Field width, of numbers, 79
Fields
    adding, in subclasses, 207
    default initialization of, 172–173
    documentation comments for, 193, 196
    final, 159, 217
    instance, 127, 150–153, 157, 174, 199
    private, 199, 206, 227
    protected, 193, 227, 282
    public, 193, 196
    public static final, 157–158, 178, 184–185
File access warning, 584
File dialogs, 545–557
    adding accessory components to, 550
fileChooser/FileIconView.java, 555
fileChooser/ImagePreviewer.java, 554
fileChooser/ImageViewerFrame.java, 552
FileContents class
    canRead, canWrite methods, 590
    getName method, 590
getInputStream, getOutputStream methods, 583, 590
getMethodName, 590
FileFilter class (Swing)
    accept method, 548, 557
    getDescription method, 548, 557
FileFilter interface (java.io package), 548
FileHandler class, 663–667, 675
    configuration parameters, 665
FileNotExtensionFilter interface, 557
FileNotFoundException, 630, 632
FileOpenService class
    openFileDialog method, 583, 590
    openMultiFileDialog method, 590
Files
    extensions of, 550
    filters for, 545
    MIME types of, 574–591
    names of, 27, 84
    opening, 545
    reading, 84
        all words from, 645
    saving, 545
    writing, 84
FileSaveService class
    saveAsFileDialog method, 590
    saveFileDialog method, 583, 590
FileView class, 549
    getIcon, getName, getDescription,
        getTypeDescription methods, 549, 557
    isTraversable method, 549, 557
fill method
    of Arrays, 116
    of Collections, 807
    of Graphics2D, 365–366, 369
Filter interface, 667
    isLoggable method, 667, 676
final access modifier, 54, 217–218
    and inner classes, 315–318
    checking, 263
    for fields in interfaces, 293
    for instance fields, 157
    for methods in superclass, 292
    for shared fields, 867–868
finalize method, 181
final clause, 640–644
    not completed normally, 680
    return statements in, 643
    unlock operation in, 851
    without catch, 642
fine, finer, finest methods (Logger), 659, 673
first method (SortedSet), 773
First Person, Inc., 11
firstKey method (SortedMap), 782
FirstSample/FirstSample.java, 46
Float class, 49, 251
    POSITIVE_INFINITY, NEGATIVE_INFINITY, NaN
    constants, 49
float type, 48
    converting to other numeric types, 60–61
Floating-point numbers, 48–49
    arithmetic computations with, 55
    converting to other numeric types, 60–61
    equality of, 97
    formatting output for, 80
    rounding, 49, 62
Floating-point overflow, 56
floor method (NavigableSet), 774
Flow layout manager, 434
FlowLayout class, 436
flush method (Handler), 674
FocusAdapter class, 422
FocusEvent class, 421
   isTemporary method, 422
FocusListener interface, 422
   focusGained, focusLost methods, 422
Font class, 370–376
   createFont method, 371
   deriveFont method, 371, 376
   getFamily, getFontName, getName methods, 376
   getLineMetrics method, 373, 376
   getStringBounds method, 372–373, 376
fontconfig.properties file, 371
font/FontTest.java, 374
FontMetrics class, 378
   getFontRenderContext method, 378
Fonts
   face names of, 369
   family names of, 369
   logical names of, 370
   size of, 370–371
   styles of, 371
   typesetting properties of, 372
"for each" loop, 108–110
   for array lists, 248
   for collections, 746, 889
   for multidimensional arrays, 118
for loop, 95–99
   comma-separated lists of expressions in, 63
   defining variables inside, 97
   for collections, 746
Forefront color, specifying, 366
darkJoin/darkJoinTest.java, 904
format, formatTo methods (String), 80
Format specifiers, 79
   formattable interface, 80
Fmtmatter class
   format, formatMessage methods, 667, 677
   getHead, getTail methods, 667, 677
   forName method (Class), 259, 263
Forte, 19
Frame class, 337, 339, 349
   getExtendedState, setExtendedState method, 349
   getIconImage method, 349
   getTitle method, 349
   isResizable method, 349
   isUndecorated, setUndecorated methods, 349
   setIconImage method, 342, 349
   setResizable method, 342, 349
   setTitle method, 342, 349
Frames
   closing by user, 341
   creating, 337, 339–342
   decorating, 342
   displaying
      information in, 350–356
      text in, 352–353
   positioning, 342–350
   properties of, 345
   size of, 345–350
Frameworks, 787
frequency method (Collections), 808
Full-screen mode, 348
Function objects, 770
Future interface, 901
   cancel method, 890, 894, 896, 916
   get method, 890, 894, 896, 916
   isCancelled, isDone methods, 890, 894, 896
   future/FutureTest.java, 891
Futures, 889–895
   FutureTask class, 889–895
   G
   Garbage collection, 66, 135
      and hash maps, 782–783
   GB18030 standard, 50
   General Public License (GPL), 14–15
   Generic programming, 697–739
      and reflection, 731–739
      and try/catch statement, 717–720
      and virtual machine, 706–712, 733–739
   clashes after type erasure in, 720–721
   classes in, 243–244, 463, 700–702
      extending/implementing other generic classes, 723
   collection interfaces in, 800
   converting to raw types, 680, 722
   debugging, 796
   expressions in, 708
   inheritance rules for, 720–723
   methods in, 702–704, 708–711, 748
   required skill levels for, 699
   type matching in, 732
   vs. arrays, 722
   vs. inheritance, 698–700
   wildcard types in, 723–731
   GenericArrayType interface, 733–734
      getGenericComponentType method, 739
   genericReflection/GenericReflectionTest.java, 734
get method
  of ArrayList, 246, 249
  of BitSet, 813
  of Future, 890, 894, 896, 916
  of GregorianCalendar, 138, 144
  of LinkedList, 758
  of List, 761, 789
  of Map, 778, 780, 789
  of PersistenceService, 591
  of Preferences, 618, 623
  of ThreadLocal, 872
  of Vector, 864
getActionCommand method
  of ActionEvent, 393, 422
  of ButtonModel, 456, 458
getActualTypeArguments method
  of ParameterizedType, 738
getAdjustable, getAdjustmentType methods
  of AdjustmentEvent, 422
getAncestorOfClass method
  of SwingUtilities, 540, 544
getApplet, getApplets methods (AppletContext), 608, 610
g.getAppletContext method (Applet), 608–610
g.getAppletInfo method (Applet), 606
gAscent, getDescent methods (LineMetrics), 377
g AudioClip method (Class), 570, 607
g getAutoCreateGaps, getAutoCreateContainerGaps methods ( GroupLayout), 514
g getAvailableFontFamilyNames, getAvailableGraphicsEnvironment methods
  of GraphicsEnvironment, 369
getBounds method (TypeVariable), 738
g getCause method (Throwable), 648
g getCenterX, getCenterY methods (RectangularShape), 359, 364
g getClass method
  and type parameters, 712
  of Class, 259
  of Object, 242
g getClass method
  of LookAndFeelInfo, 398
  of StackTraceElement, 649
getClickCount method (MouseEvent), 411, 418, 423
g getCodeBase method
  of Applet, 606–607
  of BasicService, 585, 589
g getColor method
  of Graphics, 368
  of JColorChooser, 563
g getColumns method (JTextField), 445
g getComponentPopupMenu method (JComponent), 480
g getConstructor method (Class), 275
g getConstructors method (Class), 263, 268
g getDeclaredConstructors method (Class), 264, 268
g getDeclaredField method (Class), 274
g getDeclaredFields method (Class), 264, 268, 270, 274
g getDeclaredMethods method (Class), 264, 268, 279
g getDeclaringClass method (java.lang.reflect), 268
g getDefaultScreenDevice method
  of GraphicsEnvironment, 686, 690
g getDefaultToolkit method (Toolkit), 305, 346, 349
g getDefaultUncaughtExceptionHandler method
  of Thread, 842
g getDelay method (Delayed), 880, 884
g getDescription method
  of FileFilter, 548, 557
  of FileView, 549, 557
g getDocumentBase method (Applet), 606–607
g getEnumConstants method (Class), 731–732
g getExceptionTypes method (Constructor), 268
g getExtendedState method (Frame), 349
g getFamily method (Font), 376
g getField method (Class), 274
g getFields method (Class), 263, 268, 274
g getFilter method
  of Handler, 675
  of Logger, 674
g getFirst, getLast methods
  of Deque, 775
  of List, 762
g getFirstDayOfWeek method (GregorianCalendar), 140, 144
g getFont method
  of Component, 446
  of Graphics, 377
g getFontMetrics method (JComponent), 373, 378
g getFontName method (Font), 376
g getFontRenderContext method
  of FontMetrics, 378
  of Graphics2D, 372, 377
getScrollAmount method (MouseWheelEvent), 423
getSelectedFile, getSelectedFiles methods
     (JFileChooser), 547, 556
getSelectedItem method (JComboBox), 463–464, 466
getSelectedObjects method (ItemSelectable), 456
getSelection method (ButtonGroup), 456, 458
getSequenceNumber method (LogRecord), 676
getServiceNames method (ServiceManager), 589
getShortWeekdays, getShortMonths methods
     (DateFormatSymbols), 141, 145
getSize method (Component), 348
ggetSource method (EventObject), 393, 421
ggetSourceClassName, getSourceMethodName methods
     (LogRecord), 676
getStackTrace method (Throwable), 646, 648
getState method
     of SwingWorker, 922
     of Thread, 839
getStateChange method (ItemEvent), 422
getStringBounds method (Font), 372–373, 376
getSuperclass method (Class), 242, 732
getSuppressed method (Throwable), 645, 648
getTail method (Formatter), 667, 677
getter/setter pairs. See Properties
ggetText method
     of JLabel, 447
     of JTextField, 445
getThreadID method (LogRecord), 676
getThrown method (LogRecord), 676
getTime method
     of Calendar, 217
     of GregorianCalendar, 139, 145
ggetTitle method (Frame), 349
gType method (Field), 263
gTypeDescription method (FileView), 549, 557
getTypeParameters method
     of Class, 737
     of Method, 738
getUncaughtExceptionHandler method (Thread), 842
getUpperBounds method (WildcardType), 738
getUseParentHandlers method (Logger), 674
getValue method
     of Action, 404, 410
     of AdjustmentEvent, 422
     of Map.Entry, 781
gGetWeekdays method (DateFormatSymbols), 145
gGetWheelRotation method (MouseWheelEvent), 423
gGetWidth method
     of Rectangle2D, 359
     of RectangularShape, 359, 364
ggetWindow method (WindowEvent), 423
getX, getY methods
     of MouseEvent, 411, 418, 423
     of RectangularShape, 364
getYear method (deprecated, Date), 137
GMT (Greenwich Mean Time), 136
Goetz, Brian, 820, 866
Gosling, James, 10–12
goto statement, 86, 102
Graphical applications, running, 33
Graphical User Interface (GUI)
     automatic testing, 686–690
     components of, 425, 428–563
     choice components, 451–473
     dialogs, 522–563
     menus, 451–473
     text input, 443–451
deadlocks in, 876
debugging, 635, 682–690
layout of, 433–443, 492–522
multithreading for, 827
programming, 333–381
Graphics class, 356, 378–381
copyArea method, 378, 381
drawImage method, 378, 381
drawString method, 377
getColor method, 368
g.getFont, setFont methods, 377
setColor method, 366, 368
Graphics editor application, 411–419
Graphics2D class, 356–365
draw method, 357
drawString method, 377
fill method, 365–366, 369
gGetFontRenderContext method, 372, 377
gGetPaint method, 369
setPaint method, 365, 369
GraphicsDevice class, 348, 686
GraphicsEnvironment class, 348
gGetAvailableFontFamilyNames method, 369
gGetDefaultScreenDevice method, 686, 690
gGetLocalGraphicsEnvironment method, 686, 690
Green project, 10–11
GregorianCalendar class, 136–138
     add method, 138, 144
     and Date class, 139
     constructors for, 137, 171–172
     get, set methods, 138, 144
     setFirstDayOfWeek method, 140, 144
     setTimeZone method, 139, 145
     setting date and time in, 137
Grid bag layout, 493–505
Grid layout, 439–443
GridBagConstraints class, 495
fill, anchor parameters, 497, 504–505
gridx, gridy, gridwidth, gridheight parameters, 496–498, 504
helper class for, 499–505
insets field, 497, 504–505
ipadx, ipady parameters, 505
weightx, weighty fields, 496, 504
gridbag/FontFrame.java, 500
gridbag/GBC.java, 502
GridLayout class, 436, 439–443
Group layout, 505–515
Group layout management, 493
createParallelGroup methods, 514
getAutoCreateContainerGaps, setAutoCreateContainerGaps methods, 514
getAutoCreateGaps, setAutoCreateGaps methods, 514
gerHonorsVisibility, setHonorsVisibility methods, 514
getHorizontalGroup, setVerticalGroup methods, 514
linkSize method, 514
groupLayout/FontFrame.java, 511
GridLayout, Group class, 515
GridLayout.SequentialGroup class, 515
GTK look-and-feel, 335–336
GUI. See Graphical User Interface

H
Handler class, 664
flush method, 674
getFilter, setFilter methods, 675
getFormatter method, 675
getLevel, setLevel methods, 675
publish method, 666, 674
setFormatter method, 667, 674–675
Hansen, Per Brinch, 865–866
“Has-a” relationship, 129–131
hash method (Objects), 235
Hash codes, 234–236, 763
default, 234
formatting output for, 80
Hash collisions, 764
Hash maps, 777
 concurrent, 886–888
 identity, 785–787
 linked, 783–785
setting, 778
vs. tree maps, 778
weak, 782–783
Hash sets, 763–767
adding elements to, 768
linked, 783–785
vs. tree sets, 768
Hash tables, 763–764
load factor of, 765
rehashing, 765
hashCode method, 234–236, 787
and equals method, 235
null-safe, 235
of Arrays, 236
of Object, 236, 767
of Objects, 235–236
of proxy classes, 331
of Set, 790
of String, 763
HashMap class, 777, 781
as concrete collection type, 752, 791
HashSet class, 746, 765–767
add method, 765
as concrete collection type, 752, 791
contains method, 765
Hashtable interface, 741, 791, 810–811, 888–889
as concrete collection type, 752
elements, keys methods, 811
synchronized methods, 810
hasMoreElements method (Enumeration), 746, 810–811
hasNext method
 of Interface, 745–746, 751
 of Scanner, 78
hasNextInt, hasNextDouble methods (Scanner), 78
hasPrevious method (ListIterator), 755, 761
headMap method
 of NavigableMap, 799
 of SortedMap, 794, 799
headSet method
 of NavigableSet, 794, 799
 of SortedSet, 794, 798
Heap, 776
dumping, 681
Height (in typesetting), 372
Helper classes, 499–505
Helper methods, 156, 729
Hexadecimal numbers
 formatting output for, 80
prefix for, 47
higher method (NavigableSet), 774
Hoare, Tony, 865
Hold count, 853
Holder types, 253
HotJava browser, 12, 591
Hotspot just-in-time compiler, 817
HTML (HyperText Markup Language), 13
applet element, 593, 596–599
object element, 599–600
param element, 600–606
tables, 494
target attribute, 609
HTML editors, 429
Icons
associating with arbitrary file extensions, 550
in menu items, 476–477
in sliders, 468
Identity hash maps, 785–787
identityHashCode method (System), 786–787
IdentityHashMap class, 785–787
as concrete collection type, 752
IEEE 754 specification, 49, 60
if statement, 87–90
IFC (Internet Foundation Classes), 334
IllegalAccess Exception, 270
IllegalStateException, 748, 751, 761, 775, 878
Icon class, 346
getImage method, 349, 378
image/ImageTest.java, 379
Images
accessing from applets, 606–607
displaying, 378–381
ImageViewer/ImageViewer.java, 34
Immutables classes, 157
implements keyword, 287
Implicit parameters, 152–153
none, in static methods, 159
state of, 677
import statement, 182–184
static, 184–185
importPreferences method (Preferences), 618, 623
Inconsistent state, 875
Increment operators, 56–57
Incremental linking, 7
Index (in arrays), 107
indexOf method
of List, 761
of String, 70
indexOfSubList method (Collections), 807
info method (Logger), 658–659, 673
Information hiding. See Encapsulation
Inheritance, 129–131, 203–284
and equality testing, 230–233
design hints for, 282–284
hierarchies of, 212
multiple, 212, 294
preventing, 217–218
vs. type parameters, 698, 721–723
inheritance/Employee.java, 210
inheritance/Manager.java, 211
inheritance/ManagerTest.java, 209
init method (Applet), 595–596, 600
initCause method (Throwable), 648
Initialization blocks, 176–181
static, 178
initialize method (ThreadLocal), 872
Inlining, 8, 218
Inner classes, 305–326, 391–393
accessing object state with, 307–311
anonymous, 305, 318–322, 391–392
applicability of, 312–315
defined, 305
local, 315–318
private, 309
static, 307, 322–326
syntax of, 311
innerClass/InnerClassTest.java, 309
Input maps, 407–408
Input, reading, 76–78
InputEvent class
getModifiersEx method, 412, 419
getModifiersExText method, 419
InputTest/InputTest.java, 77
Insecure certificates, 581
insert method
of JMenu, 475
of JTextArea, 923
of StringBuilder, 75
insertItemAt method (JComboBox), 464, 466
insertSeparator method (JMenu), 475
Instance fields, 127
explicit initialization of, 174
final, 157
initializing, 199
not present in interfaces, 287, 293
private, 150, 199
protected, 282
public, 150
returning values of, 153
setting values of, 154
Instance fields (cont.)
  shadowing, 151, 175
  vs. local variables, 151–152, 172
Instance variables, final, 317
instanceof operator, 63, 219–220, 292
Instances, 127
  creating on the fly, 261
int type, 47–48
  converting to other numeric types, 60–61
  fixed size for, 7
  random number generator for, 179
Integer class, 251, 253, 328
  compare method, 291
  intValue method, 254
  parseInt method, 253–254, 601
  toString method, 254
  valueOf method, 254
Integer types, 47–48
  arrays of, 238
  formatting output for, 80
  no unsigned types in Java, 48
Integrated Development Environment (IDE), 19, 28–33
  running applets in, 38, 594
Inter-applet communication, 599, 608
interface keyword, 286
Interface types, 744
Interface variables, 292
Interfaces, 286–305
  and abstract classes, 294–295
  and callbacks, 302–305
  defined, 286
  documentation comments for, 193
  extending, 293
  for collections, 741–751
  for custom algorithms, 808–809
  implementing with do-nothing methods, 400
  listener, 384–385
  marker, 298
  no instance fields in, 287, 293
  properties of, 292–294
  public, 193
  tagging, 298, 708, 789
  vs. implementations, 742–745
interfaces/Employee.java, 290
interfaces/EmployeeSortTest.java, 289
Intermittent bugs, 67, 341, 924
Internal errors, 628, 631, 655
Internal padding, 497
Internationalization, 570–571, 662
Internet Explorer
  applets in, 598
    limited Java support in, 591
    security of, 16
    versions of Java in, 9
  interrupt method (Thread), 833–836, 839
  interrupted method (Thread), 835–836
  InterruptedException, 821, 828, 833–836, 873–874, 890, 894
  IntHolder class, 253
  Intrinsics locks, 859, 866
Introduction to Algorithms (Cormen et al.), 767
intValue method (Integer), 254
Invocation handlers, 327
InvocationHandler interface, 327, 332
invoke method
  of InvocationHandler, 327, 332
  of Method, 278–282
invokeAndWait method (EventQueue), 911, 915
invokeAny, invokeAll methods (ExecutorService), 901–902
isAbstract method (Modifier), 269
isAccessible method (AccessibleObject), 274
isActionKey method (KeyEvent), 422
isCancelled, isDone methods (Future), 890, 894, 896
isDefaultButton method (JButton), 545
isDispatchThread method (EventQueue), 915
IOException, 85, 630, 632, 636, 643
"Is-a" relationship, 129–131, 213, 282
isInterrupted method (Thread), 833–836
isJavaIdentifierStart, isJavaIdentifierPart methods (Character), 52
isLocationByPlatform method (Window), 348
isLoggable method (Filter), 667, 676
isNaN method (Double), 49
isNative method (Modifier), 269
isNativeMethod method (StackTraceElement), 649
ISO 8859–1 standard, 50
isPopupTrigger method
  of JPopupMenu, 480
  of MouseEvent, 480
isPrivate method (Modifier), 263, 269
isProtected method (Modifier), 269
isProxyClass method (Proxy), 332
isPublic method (Modifier), 263, 269
isResizable method (Frame), 349
isSelected method
  of AbstractButton, 478
  of JCheckBox, 453–454
isStatic method (Modifier), 269
isStrict method (Modifier), 269
isSynchronized method (Modifier), 269
isTemporary method (FocusEvent), 422
isTraversable method (FileView), 549
isUndecorated method (Frame), 349
isVisible method (Component), 348
isWebBrowserSupported method (BasicService), 589
ItemEvent class, 421
  getItem, getItemSelectable methods, 422
  getStateChange method, 422
ItemListener interface, 422
  itemStateChanged method, 422
ItemSelectable interface, 456
  getSelectedObjects method, 456
Iterable interface, 109
Iterator interface, 745–747
  “for each” loop, 746
    generic, 748
  hasNext method, 745–746, 751
  next method, 745–746, 748, 751
  remove method, 745, 748, 751
  iterator method (Collection), 745, 750
IzPack utility, 569

J
J#, 9
J++, 9
  delegates in, 278
JApplet class, 591, 595–596
Jar Bundler utility, 569
JAR files, 190, 566–574
  and resources, 569–573
  digitally signed, 579–582
  dropping in jre/lib/ext directory, 193
  executable, 568–569
  manifest, 567–568
  sealing, 573–574
jar program, 566–567
Java
  and the Internet, 9–10
  architecture-neutral object file format of, 6
  as a dynamic language, 8
  as a programming platform, 1–2
  basic syntax of, 42–45, 145
  calling by value in, 165
case-sensitivity of, 28, 42, 51–55, 810
  command-line tools for, 25
  comments in, 45–46
  communicating with JavaScript, 599
data types in, 46–51
documentation for, 23
evolution of, 13
garbage collection in, 66, 135
  interpreter in, 7
libraries in, 13
  installing, 22–23
  misconceptions about, 13–16
  multithreading in, 8, 819–924
  networking capabilities of, 4
  no multiple inheritance in, 295
  no operator overloading in, 105
  no unsigned types in, 48
  performance of, 8, 15, 817
  portability of, 7, 55, 57
  printing to console in, 42–45
  reliability of, 5
  reserved words in, 42, 52, 55
  security of, 5–6, 16, 578–579
  short history of, 10–13
  simplicity of, 3–4
  strongly typed, 46, 288
  versions of, 12–13, 334, 493
  vs. C++, 3
Java 2 (J2), 19
Java 2D library, 356–365
  floating-point coordinates in, 357
Java bug parade, 43, 606, 661
Java collections library, 741–817
  directory tree of, 24
  vs. traditional collections libraries, 747
Java Community Process, 14
Java Concurrency in Practice (Goetz), 820
Java Development Kit (JDK), 6, 17–39
  applet viewer, 593–594
  default, 566
  documentation in, 72, 408
  downloading, 18
  fonts supplied with, 370
  installing, 18–24
  navigating directories of, 24
  setting executable path for, 20–22
Java FAQ, 16
.java file extension, 43
Java Language Specification, 43
Java look-and-feel, 406
Java Memory Model and Thread Specification, 866
Java Network Launch Protocol (JNLP), 574–591
Java Plug-in, 591, 600
java program, 26
command-line options of, 654
Java Runtime Environment (JRE), 19
Java virtual machine (JVM), 6, 11
and generic code, 706–712, 733–739
launching, 26
locales in, 140
monitoring and management of applications in, 680
optimizing execution in, 659
precomputing method tables in, 215
thread priority levels in, 840
truncating arithmetic computations in, 56
watch class loading in, 680
Java Virtual Machine Specification, 43
Java Web Start, 574–591
desktop integration warning, 577
launching, 576
printing, 585
sandbox, 577
security advisory, 583
security manager, 577
signing code in, 579–582
java.applet.Applet API, 596, 606–607, 610
java.applet.AppletContext API, 610
java.awt.BorderLayout API, 439
java.awt.Color API, 368
java.awt.Component API, 348, 355, 369, 419, 446, 494
java.awt.Container API, 389, 436
java.awt.event package, 421–423
java.awt.event.ActionEvent API, 393
java.awt.event.InputEvent API, 419
java.awt.event.MouseEvent API, 418, 480
java.awt.EventQueue API, 915
java.awt.event.WindowEvent API, 403
java.awt.event.WindowListener API, 402
java.awt.event.WindowStateListener API, 403
java.awt.FlowLayout API, 436
java.awt.Font API, 376
java.awt.font.LineMetrics API, 377
java.awt.fontMetrics API, 378
java.awt.Frame API, 349
java.awt.geom.Ellipse2D.Double API, 365
java.awt.geom.Line2D.Float API, 365
java.awt.geom.Point2D.Double API, 365
java.awt.geom.Rectangle2D.Double API, 364
java.awt.geom.Rectangle2D.Float API, 365
java.awt.geom.RectangleShape API, 364
java.awt.Graphics API, 368, 377, 381
java.awt.Graphics2D API, 369, 377
java.awt.GraphicsEnvironment API, 690
java.awt.GridLayout API, 443
java.awt.UIManager API, 520
java.awt.Robot API, 690
java.awt.Toolkit API, 348
JavaBeans, 258, 551, 603
java.beans.EventHandler API, 395
javac program, 26
current directory in, 191
javadoc program, 193–199
command-line options of, 198
comments in
class, 193–196
extracting, 198–199
field, 193, 196
general, 196
method, 193, 195–196
overview, 198
package, 193, 197
hyperlinks in, 197
links to other files in, 194
online documentation of, 199
using HTML in, 194
JavaFX, 338
java.io.Console API, 78
java.io.PrintWriter API, 85
java.lang.Class API, 242, 263, 268, 274, 573, 732, 737
java.lang.ClassLoader API, 657
java.lang.Comparable API, 290, 773
java.lang.Double API, 291
java.lang.Enum API, 257
java.lang.Exception API, 648
java.lang.Integer API, 253–254, 291
java.lang.Object API, 128, 236, 242, 767, 863
java.lang.Objects API, 236
java.lang.reflect package, 263, 274, 733–734
java.lang.reflect.AccessibleObject API, 274
java.lang.reflect.Array API, 276
java.lang.reflect.Constructor API, 263, 732
java.lang.reflect.Field API, 268, 274
java.lang.reflect.GenericArrayType API, 739
java.lang.reflect.InvocationHandler API, 332
java.lang.reflect.Method API, 268, 282, 738
java.lang.reflect.Modifier API, 269
java.lang.reflect.ParameterizedType API, 738
java.lang.reflect.Proxy API, 332
java.lang.reflect.TypeVariable API, 738
java.lang.reflect.WildcardType API, 738
java.lang.Runnable API, 833
java.lang.RuntimeException API, 649
java.lang.StackTraceElement API, 649
java.lang.String API, 70–71
java.lang.StringBuilder API, 75
java.lang.System API, 78, 616, 787
java.lang.Thread API, 826, 832, 836, 839–842
java.lang.ThreadGroup API, 843
java.lang.ThreadLocal API, 872
java.lang.Throwable API, 263, 634, 648
java.math.BigDecimal API, 106
java.math.BigInteger API, 106
java.nio.file.Paths API, 85
javap program, 312
JavaScript, 15–16
  accessing applets from, 598
  communicating with Java, 599
java.text.DateFormatSymbols API, 145
java.text.SimpleDateFormat API, 145
java.util.ArrayList API, 245–246, 249
java.util.Arrays API, 115, 233, 236, 291, 798
java.util.BitSet API, 813–817
java.util.Collection API, 750
java.util.Collections API, 797–798, 804–805, 807–808, 889
java.util.Comparator API, 773
java.util.concurrent package, 850
canned functionality classes, 905–908
efficient implementations, 866–888
java.util.concurrent.ArrayBlockingQueue API, 883
java.util.concurrent.atomic package, 868
java.util.concurrent.BlockingDeque API, 885
java.util.concurrent.BlockingQueue API, 885
java.util.concurrent.Callable API, 894
java.util.concurrent.ConcurrentHashMap API, 887–888
java.util.concurrent.ConcurrentLinkedQueue API, 887
java.util.concurrent.ConcurrentSkipListMap API, 887–888
java.util.concurrent.ConcurrentSkipListSet API, 887
java.util.concurrent.Delayed API, 884
java.util.concurrent.DelayQueue API, 884
java.util.concurrent.ExecutorCompletionService API, 902
java.util.concurrent.ExecutorService API, 900, 902
java.util.concurrent.Future API, 894
java.util.concurrent.FutureTask API, 895
java.util.concurrent.LinkedBlockingQueue API, 884
java.util.concurrent.locks.Condition API, 859, 874
java.util.concurrent.locks.Lock API, 853, 859, 874
java.util.concurrent.locks.ReentrantLock API, 853
java.util.concurrent.locks.ReentrantReadWriteLock API, 875
java.util.concurrent.PriorityBlockingQueue API, 884
java.util.concurrent.ScheduledExecutorService API, 901
java.util.concurrent.ThreadLocalRandom API, 872
java.util.concurrent.ThreadPoolExecutor API, 900
java.util.concurrent.TransferQueue API, 885
java.util.Deque API, 775
java.util Enumeration API, 811
java.util EnumMap API, 787
java.util EnumSet API, 787
java.util EventObject API, 393
java.util.GregorianCalendar API, 143
java.util.HashMap API, 781
java.util.HashSet API, 766–767
java.util.HashSet API, 766–767
java.util.Hashtable API, 811
java.util.IdentityHashMap API, 811
java.util.Iterator API, 751
java.util.LinkedHashMap API, 786
java.util.LinkedHashSet API, 786
java.util.LinkedList API, 762
java.util.List API, 761, 798
java.util.ListIterator API, 761–762
java.util/logging.ConsoleHandler API, 675
java.util/logging.ConsoleHandler API, 675
java.util/logging.FileHandler API, 675
java.util.logging.Formatter API, 676
java.util.logging.Formatter API, 677
java.util.logging.Formatter API, 677
java.util.logging.Handler API, 674–675
java.util.logging.Logger API, 673–674
java.util.logging.LogRecord API, 676
java.util.logging.LogRecord API, 676
java.util.Map API, 780–781
java.util.Map Entry API, 781
java.util.NavigableMap API, 799
java.util.NavigableSet API, 774, 799
java.util.Objects API, 233
java.util.prefs.Preferences API, 622–623
java.util.PriorityQueue API, 776
java.util.Properties API, 615, 812
setFont method, 446
setSelectionStart, setSelectionEnd methods, 924
toolTipText method, 491
jconsole program, 662, 681, 683, 868–869
jcontrol program, 594
jdb program, 691
JDialog class, 533–538
setDefaultCloseOperation method, 535, 595
setVisible method, 535, 538–539, 595
JDK. See Java Development Kit
jconsole program, 662, 681, 683, 868–869
jcontrol program, 594
jdb program, 691
JDialog class, 533–538
setDefaultCloseOperation method, 535, 595
setVisible method, 535, 538–539, 595
JDK. See Java Development Kit
 jedis text editor, 25
JEditorPane class, 450
JDialog class, 533–538
addChoosableFileFilter method, 556
getSelectedFile, getSelectedFiles methods, 547, 556
resetChoosableFilters method, 549, 556
setAcceptAllFileFilterUsed method, 549, 556
setAccessory method, 556
setCurrentDirectory method, 546, 556
setFileFilter method, 548, 556
setFileSelectionMode method, 547, 556
setFileView method, 549, 556
setMultiSelectionEnabled method, 547, 556
setSelectedFile, setSelectedFiles methods, 547, 556
showDialog, showOpenDialog, showSaveDialog methods, 540, 545, 547, 556
(JFrame class, 337, 339–343, 436
add method, 350, 355
internal structure of, 350–351
setJMenuBar method, 473–474, 476
JLabel class, 446–447, 550
setText, getIcon methods, 447
setText, setIcon methods, 446–447
JList class, 464
jmap program, 681
JMenu class
add, addSeparator methods, 474–475
insert, insertSeparator methods, 475
remove method, 475
JMenuBar class, 473–476
JMenuItem class, 476–477
setAccelerator method, 481–482
setEnabled method, 483–484
setIcon method, 476
Jmol applet, 9
JNLP API, 582–591
compiling programs with, 583
join method (Thread), 837–839
JOptionPane class, 305, 522–533
message types, 523
showConfirmDialog method, 523–525, 531
showInputDialog method, 523–524, 532–533
showInternalConfirmDialog method, 531
showInternalMessageDialog method, 531
showInternalOptionDialog method, 532
showMessageDialog method, 305, 523–524, 531
showOptionDialog method, 523–525, 531–532
JPanel class, 354, 434, 822
PasswordField class, 447–448
getPassword method, 448
setEchoChar method, 448
PopupMenu class, 479–480
isPopupTrigger method, 480
show method, 479–480
JRadioButton class, 454–458
JRadioButtonMenuItem class, 478
JRootPane class, 545
setDefaultButton method, 540, 545
JScrollBar class, 422
JScrollPane class, 451
JSlider class, 466–473
setInverted method, 469, 472
setLabelTable method, 468, 471
setMajorTickSpacing, setMinorTickSpacing methods, 472
setPaintLabels method, 468, 472
setPaintTicks method, 468, 472
setPaintTrack method, 469, 473
setSnapToTicks method, 468, 473
JSmooth classutility, 569
JSwat debugger, 691
JTextArea class, 448–449
append method, 451, 923
insert method, 451
replaceRange method, 451
setColumns, setRows methods, 448, 451
setLineWrap method, 448, 451
setTabSize method, 451
setWrapStyleWord method, 451
JTextComponent class, 444
getText method, 445
setText method, 445
JTextField class, 422, 444–446
getColumns method, 445
setColumns method, 444–445
JToolBar class, 489–490
add, addSeparator methods, 489–491
JUnit framework, 678
K

K type variable, 701
KeyAdapter class, 422
Keyboard focus, 406, 521
Keyboard mnemonics, 480–483
KeyEvent class, 421
getKeyChar, getKeyCode, getKeyModifiersText, getKeyText methods, 422
isActionKey method, 422
KeyListener interface, 422
keyPressed, keyReleased, keyTyped methods, 422
KeyPress, keyRelease methods (Robot), 690
keys method
of Hashtable, 811
of Preferences, 618, 623
keySet method (Map), 778, 781
KeyStroke class, 406
getKeyStroke method, 406, 411
Key/value pairs. See Properties
Knuth, Donald, 102
KOI-8 standard, 50

L

L suffix (long integers), 47
Labeled break statement, 103
Labels
for components, 446–447
for slider ticks, 468
Langer, Angelika, 739
last method (SortedSet), 773
lastIndexOf method
of List, 761
of String, 71
lastIndexOfSubList method (Collections), 807
lastKey method (SortedMap), 782
Launch4J utility, 569
Layout management, 433–443
absolute positioning, 516
border, 437–439
box, 493
custom, 516–520
flow, 434
grid, 439–443
grid bag, 493–505
group, 493, 505–515
sophisticated, 492–522
spring, 493
traversal order of, 521–522
LayoutManager interface, 516–520
addLayoutComponent, removeLayoutComponent methods, 520

L layoutContainer method, 520
minimumLayoutSize, preferredLayoutSize methods, 520
LayoutManager² interface, 517
Leading (in typesetting), 372
Legacy code and generics, 711–712
Legacy collections, 791, 809–817
bit sets, 813–817
enumerations, 810–811
hash tables, 810
property maps, 811–812
stacks, 812
length method
of arrays, 109
of BitSet, 813
of String, 68, 70–71
of StringBuilder, 75
Lexicographic order, 768
Library source files, installing, 22–23
Lightweight collection wrappers, 793
Line borders, 458
Line2D class, 356, 361
Line2D.Double class, 365
LineBorder class, 459, 462
Linefeed, escape sequence, 50
LineMetrics class, 373
getAscent, getDescent, getHeight, getLeading methods, 377
Lines, constructing, 361
@link comment (javadoc), 197
Linked lists, 752–762
current modifications of, 757
doubly linked, 752
printing, 759
random access in, 758, 801
LinkedBlockingDeque class, 884
LinkedBlockingQueue class, 879
LinkedHashMap class, 783–786
access vs. insertion order, 784
as concrete collection type, 752
removeEldestEntry method, 784, 786
LinkedHashSet class, 783–786
as concrete collection type, 752
LinkedList class, 744, 754, 758, 774
addFirst, addLast methods, 762
as concrete collection type, 752, 791
get method, 758
getFirst, getLast methods, 762
listIterator method, 755
nextIndex method, 758–759
previousIndex method, 759
removeAll method, 759
removeFirst, removeLast methods, 762
linkedList/LinkedListTest.java, 759
linkSize method ( GroupLayout), 514
Linux
class path in, 28
debugging applets in, 594
no thread priorities in Oracle JVM for, 840
pop-up trigger in, 479
setting paths in, 20
list interface, 789, 793, 798
add, remove methods, 761, 789
get, set methods, 761, 789
index0f, lastIndex0f method, 761
listIterator method, 761
subList method, 794
Listener interfaces, 384–385
Listener objects, 384
Listeners. See Action listeners, Event listeners,
Window listeners
ListIterator interface, 758
add method, 755–756, 761, 790
hasPrevious method, 755, 761
nextIndex, previousIndex methods, 762
previous method, 755, 762
remove method, 756
set method, 757, 761
listIterator method
of LinkedList, 755
of List, 761
Lists, 789
modifiable, 804
resizable, 804
LiveScript. See JavaScript
load method (Properties), 611, 615, 812
Local inner classes, 315–318
accessing final variables in, 315–318
Local variables
annotating, 711–712
final, 317
vs. instance fields, 151–152, 172
Locales, 140, 662
Localization, 662–663
Lock interface, 859
await method, 855–859
lock method, 853, 873–874
lockInterruptibly method, 873–874
newCondition method, 855, 859
signal method, 856–859
signalAll method, 855–859
tryLock method, 838, 873–874
unlock method, 851, 853
vs. synchronization methods, 861
Lock objects, 850–854
and inconsistent state, 875
deadlocks, 856, 868–871, 873, 875
fair, 854
hold count for, 853
intrinsic, 859, 866
not with try-with-resources statement, 851
read/write, 874–875
reentrant, 853
Lock testing and timeouts, 873–874
lockInterruptibly method (Lock), 873–874
log method (Logger), 659–660, 673
Logarithms, 60
Logger class
addHandler, removeHandler methods, 674
entering, exiting methods, 659, 673
getFilter, setFilter methods, 667, 674
getGlobal method, 658, 678
getHandlers method, 674
getLevel, setLevel methods, 674
getLogger method, 658, 673
getParent, setParent methods, 674
getUseParentHandlers, setUseParentHandlers
methods, 674
info method, 658
log method, 659–660, 673
logp method, 659, 673
logrb method, 674
setLevel method, 658
severe, warning, info, config, fine, finer, finest
methods, 659, 673
throwing method, 660, 673
Loggers
changing configuration of, 661–662
default, 658–659
hierarchical names of, 658
writing your own, 658–661
Logging, 657–677
advanced, 658–661
and resource bundles, 662
default logger for, 659
file pattern variables for, 665
file rotation for, 664
filters for, 667
formatters, 667
handlers for, 663–667
configuring, 665
including class names in, 322
Logging (cont.)
levels of, 658–659
  changing, 661–662
  localization of, 662–663
messages for, 238
  recipe for, 668–677
Logging proxy, 678
logging/LoggingImageViewer.java, 669
logging.properties file, 661–662
Logical conditions, 51
Logical “and”, “or”, 57
LogManager class, 661–662
  readConfiguration method, 661
logp method (Logger), 659, 673
logrb method (Logger), 669, 673
LogRecord class
  getLevel method, 676
  getLoggerName method, 676
  getMessage method, 676
  getMillis method, 676
  getParameters method, 676
  getResourceBundle, getResourceBundleName methods, 676
  getSequenceNumber method, 676
  getSourceClassName, getSourceMethodName methods, 676
  getThreadID method, 676
  getThrown method, 676
Long class, 251
long type, 47–48
LONG_DESCRIPTION action table name, 404
Look-and-feel, 335, 428, 493
  changing, 395–398
  pluggable, 549
LookAndFeelInfo class
  getClassName method, 398
  getName method, 398
lookup method (ServiceManager), 589
Loops
  break statements in, 102–105
  continue statements in, 103
  determinate (for), 95–99
  “for each”, 109–110
  while, 91–95
LotteryArray/LotteryArray.java, 122
LotteryDrawing/LotteryDrawing.java, 114
LotteryOdds/LotteryOdds.java, 98
lower method (NavigableSet), 774
Lowered bevel borders, 458
Low-level events, 421
Lu, Francis, 599

M
Mac OS
  executing JARs in, 569
  setting paths in, 20
main method, 44, 161–164
  body of, 44
  changing locales in, 140
  declared public static void, 43–44
  eliminating, for applets, 595
  loading classes from, 260
  not defined, 145, 178
  separate for each class, 677
  String[] args parameter of, 112–113
  tagged with throws, 85
make facility (UNIX), 149
Manager class, 204–209
  getSalary method, 206
  toString method, 237
MANIFEST.MF file, 567–568
Map interface, 788
  containsKey, containsValue methods, 781
  entrySet, keySet methods, 778, 781
  get, put methods, 778, 780, 789
  putAll method, 781
  remove method, 778
  values method, 778, 781
Map.Entry class, 781
  getKey method, 781
  getValue, setValue methods, 781
map/MapTest.java, 779
Maps, 777–782
  adding objects to, 778
  concurrent, 886–888
  garbage collecting, 782
  hash vs. tree, 778
  implementations for, 777
  keys for, 778
  enumerating, 779
  retrieving objects from, 778
  subranges of, 794
Marker interfaces, 298
Math class, 59–60
  logarithms, 60
  pow method, 59, 159
  round method, 62
  sqrt method, 59
  static constants, 60, 158–159
  trigonometric functions, 59
Matisse, 493, 505–515
Matte borders, 458
max method (Collections), 807
MAX_PRIORITY value (threads), 840
MenuListener interface, 484
  menuSelected, menuDeselected, menuCanceled
    methods, 484
menu/MenuFrame.java, 485
Menus, 473–492
  accelerators for, 481–482
  keyboard mnemonics for, 480–483
  menu bar in, 473–474
  menu items in, 473–478
    enabling/disabling, 483–487
    pop-up, 479–480
  submenus in, 473–474
merge/MenuFrame.java, 485
Menus, 473–492
  accelerators for, 481–482
  keyboard mnemonics for, 480–483
  menu bar in, 473–474
  menu items in, 473–478
    enabling/disabling, 483–487
    pop-up, 479–480
  submenus in, 473–474
Merge sort algorithm, 803
META-INF directory, 567
Metal look-and-feel, 335, 395
Method class, 263
  getDeclaringClass method, 268
  getGenericParameterTypes, getGenericReturnType
    methods, 738
  getModifiers method, 263, 268
  getName method, 263, 268
  getParameterTypes, getReturnType methods, 268
  getTypeParameters method, 738
  invoke method, 278–282
Method parameters. See Parameters
Method pointers, 278–279
Method tables, 215
Methods, 127
  abstract, 222
  accessor, 138–145, 153, 725
  adding, in subclasses, 207
  annotating, 712
  applying to objects, 133
  asynchronous, 890
  body of, 44–45
  bridge, 709–711, 721
  calling by reference vs. by value, 164–171
  casting, 218–220
  clone, 710
  concrete, 222
  consistent, 230
  deprecated, 137–138, 196, 680
  destructor, 181
  documentation comments for, 193, 195–196
  do-nothing, 400
  dynamic binding for, 209, 214–217
  exception specification in, 630
  factory, 161
  final, 215, 217–218, 263, 292
  generic, 702–704, 708–711, 748
  helper, 156, 729
  inlining, 8, 218
  invoking, 45
    arbitrary, 278–282
  mutator, 138–145, 154, 725
  names of, 201, 322
  overloading, 172
  overriding, 207, 217, 233, 283
  and exceptions, 632
  and return type, 709
  package scope of, 188
  parameterless, 45
  passing objects to, 133
  private, 156–157, 215, 263
  protected, 193, 227, 299
  public, 193, 263, 287
  reflexive, 230
  return type of, 172, 215
  signature of, 172, 215
  static, 159–160, 184–185, 215
  symmetric, 230
  tracing, 327
  transitive, 230
  varargs, 254–256, 713–714
  visibility of, in subclasses, 217
methods/MethodTableTest.java, 280
Micro Edition (Java ME), 4, 12, 19
Microsoft
  .NET platform, 6
  ActiveX, 6, 16, 600
  C#, 9, 12, 15, 218, 278, 345
  Internet Explorer, 9, 16, 591, 598
  J#, J++, 9, 278
  Visual Basic, 3, 5, 132, 345, 383, 433
  Visual Studio, 24
MIME types, 574–591
min method (Collections), 807
Minimum/maximum values, computing, 701
minimumLayoutSize method (LayoutManager), 520
MIN_PRIORITY value (threads), 840
MNEMONIC_KEY action table name, 404
mod method (BigDecimal / BigInteger), 106
Modality, 534
Model-view-controller design pattern, 428–432
  multiple views in, 430
Modifier class
  isAbstract, isInterface, isNative, isProtected,
  isStatic, isStrict, isSynchronized,
  isVolatile methods, 269
  isFinal, isPrivate, isPublic, toString methods,
  263, 269
Modulus, 55
Monitor concept, 865–866
Mosaic, 11
Mouse events, 411–419
MouseAdapter class, 415, 422
MouseEvent class, 421
getClickCount method, 411, 418, 423
getPoint method, 418, 423
getX, getY methods, 411, 418, 423
isPopupTrigger method, 480
translatePoint method, 423
MouseHandler class, 415
MouseListener interface, 413, 422
mouseClicked method, 411, 413, 415, 423
mouseDragged method, 414
mouseEntered, mouseExited methods, 415, 423
mousePressed method, 411, 413, 423
mouseReleased method, 411, 423
MouseMotionHandler class, 415
MouseMotionListener interface, 413, 415, 422
mouseDragged method, 423
mouseMoved method, 413, 415, 423
mouse/MouseComponent.java, 416
mouse/MouseFrame.java, 415
mouseMove, mousePress, mouseRelease methods
(Robot), 690
MouseWheelEvent class, 421–422
getScrollAmount method, 423
getWheelRotation method, 423
MouseWheelListener interface, 423
mouseMoved method, 423
Multidimensional arrays, 116–122
printing, 238
Multiple inheritance, 294
not supported in Java, 212
Multiple selections, 99–102
Multiplication operator, 55
multiply method (BigDecimal/BigInteger), 106
Multitasking, 819
Multithreading, 819–924
See also Threads
Mutator methods, 725
error checking in, 154

N
\n escape sequence, 50
NAME action table name, 404
NaN (not a number), 49
Napkin look-and-feel, 338
Naughton, Patrick, 10–12
NavigableMap interface, 790
subMap, headMap, tailMap methods, 799
NavigableSet interface, 771, 790, 794
ceiling, floor methods, 774
descendingIterator method, 774
higher, lower methods, 774
pollFirst, pollLast methods, 774
subSet, headSet, tailSet methods, 794, 799
nCopies method (Collections), 793, 798
Negation operator, 57
Negative infinity, 49
Nested classes. See Static inner classes
.NET platform, 6
NetBeans, 19, 25, 677
debugging in, 695
Matisse, 493, 505–515
specifying grid bag constraints in, 498
Netscape, 11
IFC library, 334
JavaScript, 16
Navigator browser, 591, 598
versions of Java in, 9
Networking, 4
new operator, 63, 74, 133, 150
return value of, 134
with arrays, 108
with generic classes, 243
with threads, 836
newCachedThreadPool method (Executors), 895–896, 899
newCondition method (Lock), 855, 859
newFixedThreadPool method (Executors), 895–896, 900
newInstance method
of Array, 275–276
of Class, 261, 263, 731–732
of Constructor, 263, 732
newProxyInstance method (Proxy), 327, 331–332
newScheduledThreadPool method (Executors), 895, 900
newSingleThreadExecutor method (Executors), 895–896, 900
newSingleThreadScheduledExecutor method (Executors), 895, 900
next method
of Interface, 745–746, 748, 751
of Scanner, 78
nextDouble method (Scanner), 76, 78
nextElement method (Enumeration), 746, 810–811
nextIndex method
of LinkedList, 758–759
of listIterator, 762
nextInt method of Random, 179
of Scanner, 76, 78
nextLine method (Scanner), 76, 78
Nimbus look-and-feel, 337
No-argument constructors, 207
NoClassDefFoundError, 28
node method (Preferences), 617, 623
noneOf method (EnumSet), 787
NORM_PRIORITY value (threads), 840
NoSuchElementException, 746, 751, 762, 775
“Not Hello World” applet, 591
Notepad text editor, 28
notHelloWorld/NotHelloWorld.java, 354
notify, notifyAll methods (Objects), 860, 863
null, 135
equality testing to, 230
NullPointerException, 629, 652
Object class, 251
Number class
factory methods, 161
parse method, 254
NumberFormatException, 651
Numeric types
casting, 61–62
converting
to other numeric types, 60–61
to strings, 253
default initialization of, 172
fixed sizes for, 7
no casting to boolean type for, 62
precision of, 105
printing, 79

O
Oak, 11
Object class, 128, 228
code method, 295–302
equals method, 228–233, 242, 795
getClass method, 242
hashCode method, 234, 236, 767
notify, notifyAll methods, 860, 863
toString method, 236, 242–242
wait method, 837–838, 860, 863
object element (HTML), 599–600
classid attribute, 600
codetype attribute, 600
Object references
as method parameters, 165
default initialization of, 172
modifying, 165
Object traversal algorithms, 786
Object variables, 223
in predefined classes, 132–136
initializing, 134
setting to null, 135
vs. C++ object pointers, 135
vs. objects, 133
Object wrappers. See Wrappers
objectAnalyzer/ObjectAnalyzer.java, 272
objectAnalyzer/ObjectAnalyzerTest.java, 272
Object-oriented programming (OOP), 4,
126–131, 203, 302
design principles of, 428
separating time measurement from calendars in, 137
vs. procedural, 126–131
Objects, 126, 128–129
analyzing at runtime, 269–274
applying methods to, 133
behavior of, 128
cloning, 294–302
comparing, 292, 768–774
concatenating with strings, 237–238
constructing, 127, 171–181
damaged, 875
default hash codes of, 234
destruction of, 181
equality testing for, 228–233, 260
finalize method of, 181
identity of, 128
implementing an interface, 292
in predefined classes, 132–136
initializing, 133
passing to methods, 133
references to, 134
serializing, 786
sorting, 286
state of, 127–128, 307–311
vs. object variables, 133
Objects class
hash method, 235
hashCode method, 235–236
Ocean look-and-feel, 335, 337
Octal numbers
formatting output for, 80
prefix for, 47
of method (EnumSet), 787
offer method
of BlockingQueue, 878–879, 885
of Queue, 775
offerFirst, offerLast methods
of BlockingDeque, 885
of Deque, 775
offsetByCodePoints method (String), 69–70
Online documentation, 70, 72, 193, 198–199
openFileDialog method (FileOpenService), 583, 590
openMultiFileDialog method (FileOpenService), 590
OpenType format, 371
Operators
arithmetic, 55–64
bitwise, 58
boolean, 57
hierarchy of, 62–63
increment/decrement, 56–57
no overloading for, 105
relational, 57
Option dialogs, 523–533
Optional operations, 797
OptionDialog/ButtonPanel.java, 529
OptionDialog/OptionDialogFrame.java, 526
or method (BitSet), 814
Oracle, 13, 19
JavaFX, 338
Ordered collections, 754, 789
ordinal method (Enum), 257
org.omg.CORBA package, 253
Originating host, 579
Output, formatting, 79–83
Output statements, 65
Overloading, 171–172
Overloading resolution, 172, 214
@override annotation, 233
overview.html, 198
Owner frame, 534
P
p (exponent), in hexadecimal numbers, 48
pack method (Window), 353, 356
pack200 compression, 566
package statement, 182, 185
package.html, 197
package-info.java, 198
Packages, 181–190
adding classes into, 185–188
default, 185
documentation comments for, 193, 197
importing, 182
names of, 259
online documentation for, 70
scope of, 188–190
sealing, 573–574
PackageTest/com/horstmann/corejava/Employee.java, 187
PackageTest/PackageTest.java, 186
paintComponent method (JComponent), 350–352, 356, 373, 378, 876
overriding, 421
pair1/PairTest1.java, 701
pair2/PairTest2.java, 705
pair3/PairTest3.java, 730
ParallelGroup class, 507, 515
ParamTest/ParamTest.java, 169
Parent classes. See Superclasses
parse method (NumberFormat), 254
parseInt method (Integer), 253–254, 601
Pascal, 10
architecture-neutral object file format of, 6
passing parameters in, 167
names of, 175
string, 45
using collection interfaces in, 809
variable number of, 254–256, 713–714
ParamTest/ParamTest.java, 169
Password fields, 447–448
PasswordChooser class, 538
Passwords
dialog box for, 539
reading from console, 77
PATH environment variable, 20
Paths class, 85
Payne, Jonathan, 12
peek method
of BlockingQueue, 878–879
of Queue, 775
of Stack, 812
peekFirst, peekLast methods (Deque), 775
Permits, 906
PersistenceService class, 585
create method, 585, 590
delete method, 590
g, getNames methods, 591
Persistent storage, 270
Phaser class, 908
Physical limitations, 627
PI constant (Math), 60, 158–159
plaf/PlafFrame.java, 396
play method (Applet), 607
Point class, 360
Point size (in typesetting), 370–371
Point2D class, 359–360
Point2D.Double class, 359, 365
Point2D.Float class, 359
poll method of BlockingQueue, 878–879, 885
of ExecutorCompletionService, 902
of Queue, 775
pollFirst, pollLast methods of Deque, 775, 885
of NavigableSet, 774
Polymorphism, 209, 213–214, 284
pop method (Stack), 812
Pop-up menus, 479–480
Pop-up triggers, 479
Pop-up windows, 579
Positive infinity, 49
PostScript Type 1 format, 371
pow method (Math), 59, 159
Precision, of numbers, 79
Preconditions, 656
Predefined action table names, 404
Predefined classes, 131–145
mutator and accessor methods in, 138–145
objects and object variables in, 132–136
Preemptive scheduling, 837
Preferences, 610–624
accessing, 617
enumerating keys in, 618
importing/exporting, 618
Preferences class, 616–624
exportSubtree, exportNode methods, 618, 623
get, put, getDataType, putDataType methods, 618, 623
importPreferences method, 618, 623
keys method, 618, 623
node, userNodeForPackage, systemNodeForPackage methods, 617, 623
systemRoot, userRoot methods, 617, 622
preferences/PreferencesTest.java, 619
preferredLayoutSize method (LayoutManager), 520
previous method (ListIterator), 755, 762
previousIndex method
of LinkedList, 759
of ListIterator, 762
Prime numbers, 814
Primitive types, 46–51, 228
and type parameters, 712
as method parameters, 165
converting to objects, 251
final fields of, 157
Princeton University, 5
print method (System.out), 45, 79
printf method (System.out), 79–83
conversion characters for, 79
flags for, 81
for date and time, 81–82
parameters of, 254
println method (System.out), 45, 76, 657
printStackTrace method (Throwable), 262–263, 646, 678
PrintStream class, 583
PrintWriter class, 84–85
Priority queues, 776
vs. tree sets, 776
PriorityBlockingQueue class, 879, 884
PriorityQueue class, 776
as concrete collection type, 752, 791
priorityQueue/PriorityQueueTest.java, 777
private access modifier, 150, 157–164, 227, 309
checking, 263
for fields, in superclasses, 206
for methods, 156–157
Procedures, 126
process method (SwingWorker), 915–918, 922
Processes, vs. threads, 820
Producer threads, 878
Profilers, 682
Programs. See Applications
Properties class, 611–616, 791, 810–812
getProperty method, 612, 615, 812
load, store methods, 611, 615, 812
properties/PropertiesTest.java, 613
Property maps, 345, 611–616, 811–812
comments in, 395
names of, 611
reading/writing, 611
PropertyChangeListener, 551
protected access modifier, 227, 299
for fields, 282
Proxies, 326–332
properties of, 331–332
purposes of, 327
Proxy class, 331–332
getProxyClass, isProxyClass methods, 331–332
newProxyInstance method, 327, 331–332
ProxyTest.java, 329
public access modifier, 42, 55, 149–150, 157–164, 287
checking, 263
for fields in interfaces, 293
for main method, 43
for only one class in source file, 148
not specified for interfaces, 286
publish method
- of Handler, 666, 674
- of SwingWorker, 915, 917, 922
Pure virtual functions (C++), 223
push method (Stack), 812
put method
- of BlockingQueue, 878–879, 885
- of Map, 778, 780, 789
- of Preferences, 618, 623
putAll method (Map), 781
putDataType methods (Preferences), 618, 623
putFirst, putLast methods (BlockingDeque), 885
putIfAbsent method (concurrent collections), 887–888
putValue method (Action), 404, 410

Q
Queue interface, 743–744, 774–776
- add, element, offer, peek, poll, remove methods, 775
Queues, 742–745, 774–776
- blocking, 878–879
- concurrent, 886–888
- double-ended. See Deques
QuickSort algorithm, 113, 803

R
\r escape sequence, 50
Race conditions, 843–849
Radio buttons, 454–458
- in menus, 477–478
radioButton/RadioButtonFrame.java, 456
Ragged arrays, 120–123
Raised bevel borders, 458
Random class, 179
- nextint method, 179
- thread-safe, 872
Random number generation, 179, 872
RandomAccess interface, 789, 804, 806
- range method (EnumSet), 787
Raw types, 706, 712–713
- converting parameterized types to, 722
readConfiguration method (LogManager), 661
readLine, readPassword methods (Console), 78
readLock method (ReentrantReadWriteLock), 873–874
Read/write locks, 874–875
Rectangle class, 360, 771
Rectangle2D class, 356, 358, 360–361
- getWidth method, 359
- setRect method, 359
Rectangle2D.Double class, 358–359, 364
Rectangle2D.Float class, 358–359, 365
Rectangles
- comparing, 771
- drawing, 357
RectangularShape class, 359
- getWidth, getHeight, getCenterX, getCenterY methods, 359
- getX, getY, getWidth, getHeight, getCenterX, getCenterY, getMinX, getMaxX, getMinY, getMaxY methods, 364
Recursive computations, 903
RecursiveAction class, 903
RecursiveTask class, 903
Red-black trees, 767
Redundant keywords, 293
Reentrant locks, 853
ReentrantLock class, 850–854
ReentrantReadWriteLock class, 874–875
- readLock, writeLock methods, 873–874
Reflection, 204, 258–282
- analyzing	xx classes, 263–269
- objects, at runtime, 269–274
- and generics, 274–278, 716, 731–739
- overusing, 284
- reflection/ReflectionTest.java, 265
Relational operators, 57
- precedence of, 63
Relative resource name, 570
remove method
- of ArrayList, 247, 249
- of BlockingQueue, 878–879
- of Collection, 749–750
- of concurrent collections, 887–888
- of Interface, 745, 751
- of Iterator, 748
- of JMenu, 475
- of List, 761, 789
- of ListIterator, 756
- of Map, 778
- of Queue, 775
- of ThreadLocal, 872
removeAll method
  of Collection, 749–750
  of LinkedList, 759
removeEldestEntry method (LinkedHashMap), 784, 786
removeFirst, removeLast methods
  of Deque, 775
  of LinkedList, 762
removeHandler method (Logger), 674
removeItem, removeItemAt, removeAllItems methods
  (JComboBox), 464, 466
removeLayoutComponent method (LayoutManager), 520
removePropertyChangeListener method (Action), 404–405
repaint method
  of Component, 352
  of JComponent, 355, 821, 923
replace method
  of concurrent collections, 887–888
  of String, 71
replaceAll method (Collections), 807
replaceAll method (Collections), 807
reverse method (Collections), 808
reverseOrder method (Collections), 803, 805
Robot class, 686–690
  createScreenCapture method, 690
delay method, 690
  keyPress, keyRelease, mouseMove, mousePress, mouseRelease methods, 690
robot/RobotTest.java, 687
rotate method (Collections), 808
round method (Math), 62
Rounding mode, 106
RoundingMode class, 106
run method (TransferRunnable)
  of Runnable, 833
  of Thread, 829, 832
runFinalizersOnExit method (System), 181
Runnable interface, 827
run method, 833
Runtime
  adding shutdown hooks, 181
type identification at, 259
type inquiries, 712–713
RuntimeException, 628–629, 649, 651

S
@SafeVarargs annotation, 714
Sandbox, 578–579
saveAsFileDialog method (FileSaveService), 590
saveFileDialog method (FileSaveService), 583, 590
Scanner class, 76–78, 84–85
  next, hasNext, hasNextInt, hasNextDouble methods, 78
  nextInt, nextDouble, nextLine methods, 76, 78
Scheduled execution, 900–901
ScheduledExecutorService class, 900
  schedule, scheduleAtFixedRate, scheduleWithFixedDelay methods, 901
Scroll panes, 448–451
Scrollbars, 449–451
Sealing, 573–574
Secure certificates, 580
Security, 5–6, 16, 578–579
@see comment (javadoc), 196
Self-signed certificates, 582
Semantic events, 421
Semaphore class, 906
SequentialGroup class, 507, 515
Serialization, 786
  of applet objects, 598
ServiceManager interface, 583
  getServiceNames method, 589
  lookup method, 589
ServletException, 639
Servlets, 639
Set interface, 790
  add, equals, hashCode methods, 790
set method
  of ArrayList, 246, 249
  of BitSet, 813
  of GregorianCalendar, 138, 144
  of List, 761
  of ListIterator, 757, 761
  of ThreadLocal, 872
  of Vector, 864
setAccelerator method (JMenuItem), 481–482
setAcceptAllFileFilterUsed method (JFileChooser), 549, 556
setAccessible method (AccessibleObject), 270, 274
setAccessory method (JFileChooser), 556
setAction method (AbstractButton), 476
setActionCommand method (AbstractButton), 458
setAutoCreateGaps, setAutoCreateContainerGaps methods (GroupLayout), 514
setBackground method (Component), 366, 369
setBorder method (JComponent), 459, 462
setBounds method (Component), 342, 348, 516
  coordinates in, 344
setCharAt method (StringBuilder), 75
setClassAssertionStatus method (ClassLoader), 657
setColor method (Graphics), 366, 368
  of JColorChooser, 563
setColumns method
  of JTextArea, 448, 451
  of JTextField, 444–445
setComponentPopupMenu method (JComponent), 479–480
setCurrentDirectory method (JFileChooser), 546, 556
setCursor method (Component), 419
setDaemon method (Thread), 841
setDebugGraphicsOptions method (Component), 682–683
setDefault method (Locale), 140
setDefaultAssertionStatus method (ClassLoader), 657
setDefaultButton method (JRootPane), 540, 545
setDefaultCloseOperation method (JDialog), 535, 595
setDefaultUncaughtExceptionHandler method (Thread), 679, 842
setDisplayMnemonicIndex method (AbstractButton), 481, 483
setEchoChar method (JPasswordField), 448
setEditable method (JComboBox), 463, 466
setEnabled method
  of Action, 404, 410
  of JMenuItem, 483–484
setExtendedState method (Frame), 349
setFileFilter method (JFilechooser), 548, 556
setFileSelectionMode method (JFileChooser), 547, 556
setFileView method (JFileChooser), 549–550, 556
setFilter method
  of Handler, 675
  of Logger, 667, 674
setFont method
  of Graphics, 377
  of JComponent, 446
setForeground method (Component), 366, 369
setFormatter method (Handler), 675
  of Logger, 658, 674
setFrameFromCenter method (Ellipse2D), 361
setFrameFromDiagonal method (Ellipse2D), 360
setHonorsVisibility method (GroupLayout), 514
setHorizontalAlignment method (AbstractButton), 476–477
setIcon method
  of JLabel, 446–447
  of JMenuItem, 476
setIconImage method (Frame), 342, 349
setInheritsPopupMenu method (JComponent), 479–480
setInverted method (JSlider), 469, 472
setMenuBar method (JFrame), 473–474, 476
setLabelTable method (JSlider), 468, 472, 711
setLayout method (Container), 436
setLevel method
  of Handler, 675
  of Logger, 658, 674
setLineWrap method (JTextArea), 448, 451
setLocation method (Component), 342, 344, 348
  coordinates in, 344
setLocationByPlatform method (Window), 348
setLookAndFeel method (UIManager), 396, 398
setMajorTickSpacing, setMinorTickSpacing methods (JSlider), 472
setMnemonic method (AbstractButton), 481, 483
setModel method (JComboBox), 464
setMultiSelectionEnabled method (JFileChooser), 547, 556
setOut method (System), 159
setPackageAssertionStatus method (ClassLoader), 657
setPaint method (Graphics2D), 365, 369
setPaintLabels method (JSlider), 468, 472
setPaintTicks method (JSlider), 468–469, 472
setPaintTrack method (JSlider), 473
setParent method (Logger), 674
setPriority method (Thread), 840
setProperty method (System), 661
setRect method (Rectangle2D), 359
setResizable method (Frame), 342, 349
setRows method (JTextArea), 448, 451
Sets, 765
  concurrent, 886–888
  elements of
  mutating, 765
  intersection of, 799
  subranges of, 794
setSelected method
  of AbstractButton, 478
  of JCheckBox, 452, 454
setSelectedFile, setSelectedFiles methods
  (JFileChooser), 547, 556
setSelectionStart, setSelectionEnd methods
  (JComponent), 924
set/SetTest.java, 766
setSize method (Component), 348
setSnapToDevice ticks on, 467–468
vertical, 466
setTime method
  of Calendar, 217
  of GregorianCalendar, 139, 145
setTitle method (JFrame), 342, 349
setToolTipText method (JComponent), 491
setUseParentHandlers method (Logger), 674
setValue method
  of Map.Entry, 781
setVerticalGroup method (GroupLayout), 514
setVisible method
  of Component, 341–342, 348, 923
  of JDialog, 535, 538–539, 595
setWrapStyleWord method (JTextArea), 451
severe method (Logger), 659, 673
Shallow copies, 295, 298
Shape interface, 356–357
Shell
  redirection syntax of, 85
  scripts in, 192
Shift operators, 58
short type, 47
Short class, 251
SHORT_DESCRIPTION action table name, 404
show method (JPopupMenu), 479–480
showConfirmDialog method (JOptionPane), 523–525, 531
showDialog method
  of JColorChooser, 563
  of JFileChooser, 540, 545, 547, 556
showDocument method
  of AppletContext, 609–610
  of BasicService, 589
showInputDialog method (JOptionPane), 523–524, 532–533
showInternalConfirmDialog/showInternalMessageDialog methods (JOptionPane), 531
showInternalOptionDialog method (JOptionPane), 532
showMessageDialog method (JOptionPane), 305, 523–525, 531–532
showStatus method (Applet), 609–610
shuffle method (Collections), 804–805
shuffle/ShuffleTest.java, 804
Shuffling, 804
Shutdown hooks, 181
shutdown method (ExecutorService), 896, 900
shutdownNow method (ExecutorService), 896, 901
“Sieve of Eratosthenes” benchmark, 814–817
sieve/sieve.cpp, 815
sieve/Sieve.java, 814
signal method (Condition), 856–859, 869
signalAll method (Condition), 855–859, 869
Signatures (of methods), 172, 215
Signed code, 579–582
singleframe/SingleFrameTest.java, 340
Single-thread rule (Swing), 911, 923–924
singleton, singletonList, singletonMap methods
  (Collections), 793, 798
size method
  of ArrayList, 245–246
  of Collection, 749–750
  of concurrent collections, 886
sizedFrame/SizedFrameTest.java, 347
sleep method (Thread), 821, 826, 828, 833
Sliders, 466–473
ticks on, 467–468
vertical, 466
slider/SliderFrame.java, 469
SMALL_ICON action table name, 404
SoftBevelBorder class, 459, 462
Software Development Kit (SDK), 19
Solaris
  executing JARs in, 569
  setting paths in, 20
sort method
  of Arrays, 113, 116, 286, 289, 291
  of Collections, 802–805
SortedMap interface, 790
  comparator method, 782
  firstKey, lastKey methods, 782
  subMap, headMap, tailMap methods, 794, 799
SortedSet interface, 790, 794
  comparator method, 773
  first, last methods, 773
  subSet, headSet, tailSet methods, 794, 798
Sorting
  algorithms for, 113, 802–805
  arrays, 113–116, 289
  assertions for, 655
  in reverse order, 803
Special characters, 50
Splash screen, 260
Spring layout, 493
sqrt method (Math), 59
src directory, 24
src.zip file, 22
Stack interface, 741, 791, 810, 812
  peek, pop, push methods, 812
Stack trace, 646–649, 868
Stacks, 812
StackTraceElement class, 646
  getClassName, getMethodName methods, 649
  getLineNumber method, 649
  isNativeMethod method, 649
  toString method, 646, 649
StackTrace/StackTraceTest.java, 647
Standard Edition (Java SE), 12, 19
Standard Java library
  online API documentation for, 70, 72, 193, 198
  packages in, 182
Standard Template Library (STL), 742, 747
start method
  of Applet, 596
  of Thread, 829, 832, 836
  of Timer, 305
startsWith method (String), 71
stateChanged method (ChangeListener), 466–467
Statements, 45
static access modifier, 157–164
  for fields in interfaces, 293
  for main method, 44
Static binding, 215
Static constants, 158–159
  documentation comments for, 196
Static fields, 157–158
  accessing, in static methods, 160
  importing, 184–185
  initializing, 178
static final access modifier, 54
Static imports, 184–185
Static inner classes, 307, 322–326
Static methods, 159–160
  accessing static fields in, 160
  importing, 184–185
  names of, 322
Static variables, 158
  final, 317
staticInnerClass/StaticInnerClassTest.java, 324
STATICTest/STATICTest.java, 163
stop method
  of Applet, 596
  of Thread class (deprecated), 833, 839, 875–877
  of Timer, 305
store method (Properties), 611, 615, 812
Strategy design pattern, 427
StreamHandler class, 664, 666
Streams, 583
strictfp keyword, 56
StrictMath class, 59–60
String class, 64–75
  charAt, codePointAt methods, 69–70
  codePointCount method, 68, 71
  compareTo method, 70
  endsWith method, 70
  equals method, 67, 70
  equalsIgnoreCase method, 67
  format, formatTo methods, 80
  hashCode method, 234, 763
  immutability of, 65, 157, 218
  index0f method, 70, 172
  lastIndexOf method, 71
  length method, 68, 70–71
  offsetByCodePoints method, 69–70
  replace method, 71
  startsWith method, 71
  substring method, 64, 71, 794
  toLowerCase, toUpperCase methods, 71
  trim method, 71, 445
StringBuilder class, 74–75
append method, 74–75
appendCodePoint method, 75
delete method, 75
insert method, 75
length method, 75
setCharAt method, 75
toString method, 75
Strings, 64–75
building, 74–75
code points/code units of, 68
concatenating, 64–65
with objects, 237–238
converting to numbers, 253
empty, 68
equality testing, 67
formatting output for, 79–83
immutability of, 65
length of, 64, 68
null, 68
shared, in compiler, 65, 67
substrings of, 64
using "..." for, 45
Strongly typed languages, 46
Subclasses, 204–227
adding fields/methods to, 207
anonymous, 321
cloning, 299
comparing objects from, 292
method visibility in, 217
no access to private fields of superclass, 227
overriding superclass methods in, 207
subList method (List), 794
subMap method
of NavigableMap, 799
of SortedMap, 794, 799
Submenus, 473–474
submit method
of ExecutorCompletionService, 900, 902
of ExecutorService, 896
Subranges, 794
subSet method
of NavigableSet, 794, 799
of SortedSet, 794, 798
Substitution principle, 213
substring method (String), 64, 71, 794
subtract method (BigDecimal / BigInteger), 106
Subtraction operator, 55
Sun Java Studio, 19
Sun Microsystems, 2, 5–13, 15, 334
HotJava browser, 12, 591
Sun ONE Studio, 19
super keyword, 206, 298, 725–726
vs. this, 207–208
Superclasses, 204–227
accessing private fields of, 206
common operation and fields in, 282
moving common fields and methods to, 222
overriding methods of, 233
throws specifiers in, 632, 637
Supertype bounds, 725–728
Supplementary characters, 51
@SuppressWarnings annotation, 101, 251, 711–712, 714, 718, 720
Surrogates area, 51
suspend method (deprecated, Thread), 839, 875–877
swap method (Collections), 807
Swing, 333–381, 425–563
advantages of, 335
and threads, 909–915, 923–924
debugging, 682–690
executing in full-screen mode, 348
implementing applets with, 591–596
model-view-controller analysis of, 432–433
starting program, 341
Swing graphics debugger, 682
SwingConstants interface, 293, 446
swing.properties file, 395
swing/SwingThreadTest.java, 912
SwingUtilities class
gerAncestorOfClass method, 540, 544
updateComponentTreeUI method, 396
SwingWorker class, 915–923
doInBackground method, 915, 917, 922
execute method, 916, 922
getState method, 922
process method, 915–918, 922
publish method, 915, 917, 922
swingWorker/SwingWorkerTest.java, 918
switch statement, 99–102
enumerated constants in, 102
missing break statements in, 680
SWT toolkit, 338
synch2/Bank.java, 862
synch/Bank.java, 857
Synchronization, 843–877
condition objects, 854
in Vector, 763
lock objects, 850–854
lock testing and timeouts, 873–874
monitor concept, 865–866
Synchronization (cont.)
  race conditions, 843–849
  read/write locks, 874
  volatile fields, 866–867
Synchronization primitives, 907
Synchronization wrappers, 888–889
Synchronized blocks, 864–865
synchronized keyword, 850, 859–863
Synchronized views, 796
synchronizedCollection methods (Collections), 796, 798, 889
Synchronizers, 905–908
  barriers, 907–908
  countdown latches, 907
  exchangers, 908
  semaphores, 906
  synchronous queues, 908
SynchronousQueue class, 906–908
Synth look-and-feel, 337
System class
  console method, 78
  exit method, 44
  getProperties method, 611–612, 616
  getProperty method, 616
  identityHashCode method, 786–787
  out constant, 159
  runFinalizersOnExit method, 181
  setOut method, 159
  setProperty method, 661
System of Patterns, A (Buschmann et al.), 428
SystemColor class, 367–368
System.err class, 679
System.in class, 76
systemNodeForPackage method (Preferences), 617, 623
System.out class, 45, 159, 679
  print method, 79
  printf method, 79–83, 254
  println method, 76, 657
systemRoot method (Preferences), 617, 622
tailSet method
  of NavigableSet, 794, 799
  of SortedSet, 794, 798
take method
  of BlockingQueue, 878–879, 885
  of ExecutorCompletionService, 902
takeFirst, takeLast methods (BlockingDeque), 885
TalkingClock class, 307–317
tar command, 566
target attribute (HTML), 609
Tasks
decoupling from mechanism of running, 829
  groups of, controlling, 901–902
  interrupting, 822
  multiple, 819
  running asynchronously, 890
time-consuming, 910–915
  work stealing for, 904
Template code bloat, 707
Text
  centering, 372
  displaying, 352–353
  special fonts for, 369
  typesetting properties of, 372
Text areas, 448–449
  formatted text in, 450
  scrollbars in, 449–451
Text fields, 444–446
  blank, 445
  columns in, 444
  preferred size of, 444
Text input, 443–451
  labels for, 446–447
  password fields, 447–448
  scroll panes, 448
  text areas, 448–449
  text fields, 444–446
TextPad text editor, 25
text/TextComponentFrame.java, 450
this keyword, 152, 175
  in first statement of constructor, 176
  in inner classes, 311
  vs. super, 207–208
Thread class, 832
currentThread method, 833–836
  extending, 829
  getDefaultUncaughtExceptionHandler method, 842
getState method, 839
getUncaughtExceptionHandler method, 842
interrupt method, 833–836, 839
interrupted method, 835–836
isInterrupted method, 833–836
join method, 837–839
methods with timeout, 838
resumes method, 839
run method, 829, 832
setDaemon method, 841
setDefaultUncaughtExceptionHandler method, 679, 842
setPriority method, 840
setUncaughtExceptionHandler method, 842
sleep method, 821, 826, 828, 833
start method, 829, 832, 836
stop method (deprecated), 833, 839, 875–877
suspend method (deprecated), 839, 875–877
yield method, 841
Thread dump, 869
Thread groups, 842
Thread pools, 895–900
of fixed size, 896
ThreadDeath error, 839, 843, 876
ThreadGroup class, 842
uncaughtException method, 842–843
ThreadLocal class, 872
get, set, initialize, remove methods, 872
ThreadLocalRandom class, 872
current method, 872
ThreadPoolExecutor class, 896
getLargestPoolSize method, 900
threadPool/ThreadPoolTest.java, 897
Threads, 8
and Swing, 909–915, 923–924
blocked, 833, 837–839
daemon, 841
defined, 820–833
handlers for uncaught exceptions in, 841–843
interrupting, 833–836
new, 836
preemptive scheduling for, 837
priorities of, 840
producer/customer, 878
properties of, 839–843
purposes of, 827–833
runnable, 836–837
simple procedure for, 827–833
states of, 836–839
terminated, 833, 839
thread-local variables in, 871–872
timed waiting, 837–839
unblocking, 856
vs. processes, 820
waiting, 837–839
work stealing for, 904
Thread-safe collections, 885–888
callables and futures, 889–905
concurrent, 886–888
copy on write arrays, 888
synchronization wrappers, 888–889
Thread.UncaughtExceptionHandler interface, 841–843
throw keyword, 632–633
Throwable class, 263, 628, 651
addSuppressed, getSuppressed methods, 645, 648
getMessage method, 634
getStackTrace method, 646, 648
initCause, getCause methods, 648
printStackTrace method, 262–263, 646, 678
toString method, 634
throwing method (Logger), 660, 673
throws keyword, 630–632
for main method, 85
@throws comment (javadoc), 195
Ticks, 467
icons for, 468
labeling, 468
snapping to, 468
Time measurement vs. calendars, 137
Timed waiting threads, 837–839
Timeless Way of Building, The (Alexander), 426
TimeoutException, 890
TimePrinter class, 307–317
actionPerformed method, 314, 316
timer/TimerTest.java, 304
toArray method
of ArrayList, 716
of Collection, 247, 749–750
toBack, toFront methods (Window), 348
toLowerCase, toUpperCase methods (String), 71
Tomcat, 574–591
Toolbars, 488–490
detached, 489
dragging, 488
title of, 490
vertical, 490
toolBar/ToolBarTest.java, 491
Toolkit class
createCustomCursor method, 414, 419
defaultToolkit method, 305, 346, 349
getScreenSize method, 346, 349
Toolkit-modal dialogs, 534
ToolTips, 490–492
toString method
  adding to all classes, 239
  and Formattable, 80
  of Arrays, 110, 115
  of Date, 133
  of Employee, 236–237
  of Enum, 256–257
  of Integer, 254
  of Manager, 237
  of Modifier, 263, 269
  of Object, 236, 242–242
  of proxy classes, 331
  of StackTraceElement, 646, 649
  of StringBuilder, 75
  of Throwable, 634
  working with any class, 270–271
Total ordering, 770
TraceHandler class, 327
Tracing execution flow, 659
transfer, tryTransfer methods (TransferQueue), 885
TransferQueue interface, 880
  transfer, tryTransfer methods, 885
  translatePoint method (MouseEvent), 423
Traversal order, 521–522
Tree sets, 767–768
  adding elements to, 768, 770
  red-black, 767
  total ordering of, 770
  vs. hash sets, 768
  vs. priority queues, 776
TreeMap class, 777, 782, 790
  as concrete collection type, 752, 791
  vs. HashMap, 778
TreeSet class, 767–768, 774, 790
  addAll method, 770
  as concrete collection type, 752, 791
treeSet/Item.java, 772
treeSet/TreeSetTest.java, 771
Trigonometric functions, 59
trim method (String), 71, 445
trimToSize method (ArrayList), 245–246
Troubleshooting, 27–28
TrueType format, 371
try/catch statement, 262, 635–640
  and generics, 717–720
  decoupling, 642
  wropping entire task in try block, 651
try/finally statement, 640–644
  decoupling, 642
tryLock method (Lock), 838, 873–874
Try-with-resources statement, 644–646
  no locks with, 851
Two-dimensional arrays, 116–122
Type interface, 733–734
Type erasure, 708–712, 716
  clashes after, 720–721
Type parameters, 243
  and arrays, 713
  and primitive types, 712
  converting to raw types, 722
  vs. inheritance, 698
Type variables
  bounds for, 704–706
  in exceptions, 718
  matching in generic methods, 732
  names of, 701
  no instantiating for, 715–716
Typesetting terms, 372
TypeVariable interface, 733–734
  getBounds, getName methods, 738
U
UCSD Pascal system, 6
UIManager class, 398
  setLookAndFeel method, 396, 398
UML (Unified Modeling Language) notation, 130–131
UnavailableServiceException, 583
uncaughtException method
  of ThreadGroup, 842–843
  of UncaughtExceptionHandler, 841–843
UncaughtExceptionHandler interface, 841–843
Unchecked exceptions, 261–262, 629–631
  applicability of, 651
Unequality operator, 57
Unicode standard, 7, 50, 64
  in char type, 49
Unit testing, 162
University of Illinois, 11
UNIX
  JNLP configuration, 577
  setting paths, 20, 28, 190–192
  system directories, 611
troubleshooting, 28
unlock method (Lock), 851, 853
Unmodifiable views, 794–796
unmodifiableCollection methods (Collections), 795, 797
UnsupportedOperationException, 779, 793, 795, 797
unsynch/Bank.java, 846
unsynch/TransferRunnable.java, 847
unsynch/UnsynchBankTest.java, 845
updateComponentTreeUI method (SwingUtilities), 396
User input, 445
errors of, 627
User Interface. See Graphical User Interface
userNodeForPackage method (Preferences), 617, 623
userRoot method (Preferences), 617, 622
“Uses-a” relationship, 129–131
UTC (Coordinated Universal Time), 136

V
V type variable, 701
validate method (Component), 446, 923
valueOf method
of BigDecimal/BigInteger, 105–106
of Enum, 256–257
of Integer, 254
values method (Enum), 257, 778, 781
Varargs, 254–256
and type parameters, 713–714
Variables, 51–55
blank final, 317
copying, 295
declarations of, 52
deprecated, 196
final, accessing from outer methods, 315–318
initializing, 53, 199
instance, 317
local, 317, 711
names of, 51–55
package scope of, 188
printing/logging values of, 677
static, 158, 317
Vector class, 741, 789, 791, 810, 864–865, 888–889
elements method, 811
for dynamic arrays, 244
get, set methods, 864
synchronization in, 763
@version comment (javadoc), 196, 198
Views, 428, 792
bulk operations for, 800
checked, 796
restricted, 797
subranges of, 794
synchronized, 796
unmodifiable, 794–796
Visual Basic
built-in date type in, 132
event handling in, 383
forms in, 433
properties in, 345
reliability of, 5
syntax of, 3
Void class, 251
void keyword, 44
Volatile fields, 866–867
volatile keyword, 867–868
von der Ahé, Peter, 703

W
wait method (Object), 837–838, 860, 863
Wait sets, 855
warning method (Logger), 659, 673
Weak references, 783
WeakHashMap class, 782–783, 786
as concrete collection type, 752
Weakly consistent iterators, 886
WeakReference object, 783
webstart/CalculatorFrame.java, 586
WelcomeApplet/WelcomeApplet.html, 37
WelcomeApplet/WelcomeApplet.java, 38
Welcome/Welcome.java, 27
WHEN_ANCESTOR_OF_FOCUSED_COMPONENT map, 407–408
WHEN_FOCUSED map, 407
WHEN_IN_FOCUSED_WINDOW map, 407
while loop, 91–95
White Paper, 2–9
Whitespace, in compiler, 44
Wildcard types, 699, 723–731
arrays of, 713
capturing, 728–731
supertype bounds for, 725–728
unbounded, 728
WildcardType interface, 733–734
getUpperBounds, getLowerBounds methods, 738
Window class, 423
isLocationByPlatform, setLocationByPlatform
methods, 348
pack method, 353, 356
toBack, toFront methods, 348
Window listeners, 399–403
WindowAdapter class, 422
WindowClosing event, 482
WindowEvent class, 384, 399, 421
getNewState, getOldState methods, 403, 423
getWindow, getOppositeWindow, getScrollAmount
method, 423
WindowFocusListener interface, 422
windowGainedFocus, windowLostFocus methods, 423
WindowListener interface, 422
  windowActivated, windowDeactivated methods, 399, 402, 423
  windowClosing, windowClosed methods, 399–402, 423
  windowIconified, windowDeiconified methods, 399, 402, 423
  windowOpened method, 399, 402, 423
Windows. See Dialogs
Windows operating system
  Alt+F4 in, 482
  class path in, 28, 190, 192
  debugging applets in, 594
  default installation location in, 20
  default location in, 664
  device context in, 351
  executing JARs in, 569
  file separators in, 571
  fonts shipped with, 370
  look-and-feel, 335–336
  pop-up trigger in, 479
  registry in, 618
  resources in, 570
  setting paths in, 20
  shell in, 22
  thread priority levels in, 840
WindowStateListener interface, 399, 422
  windowStateChanged method, 403, 423
Wirth, Niklaus, 6, 10–11, 126
Work stealing, 904
Wrappers, 251–254
  equality testing for, 252
  immutability of, 251
  lightweight collection, 793
writeLock method (ReentrantReadWriteLock), 873–874
X
  X11 programming, 351
  XML (Extensible Markup Language), 13–14
  xor method (BitSet), 814
Y
  yield method (Thread), 841
Z
  ZIP format, 190, 566