

Rogers Cadenhead

**Seventh
Edition**

**Covers
Java 8 and
Android
Development**

Sams **Teach Yourself**

JavaTM

in **21 Days**



SAMS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Rogers Cadenhead

Sams **Teach Yourself**

JavaTM

in **21 Days**

SAMS

800 East 96th Street, Indianapolis, Indiana 46240

Sams Teach Yourself Java™ in 21 Days

Copyright © 2016 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-672-33710-9

ISBN-10: 0-672-33710-X

Library of Congress Control Number: 2015916217

Printed in the United States of America

First Printing December 2015

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Special Sales

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

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

For questions about sales outside the U.S., please contact international@pearsoned.com.

Editor-in-Chief

Mark Taub

Executive Editor

Mark Taber

Managing Editor

Kristy Hart

Project Editor

Elaine Wiley

Copy Editor

Barbara Hacha

Senior Indexer

Cheryl Lenser

Proofreader

Chuck Hutchinson

Technical Editor

Boris Minkin

Editorial Assistant

Vanessa Evans

Cover Designer

Mark Shirar

Senior Compositor

Gloria Schurick

Contents at a Glance

Introduction	1
WEEK I: The Java Language	7
DAY 1 Getting Started with Java	9
DAY 2 The ABCs of Programming	37
DAY 3 Working with Objects	67
DAY 4 Lists, Logic, and Loops	95
DAY 5 Creating Classes and Methods	125
DAY 6 Packages, Interfaces, and Other Class Features	157
DAY 7 Exceptions and Threads	191
WEEK II: The Java Class Library	223
DAY 8 Data Structures	225
DAY 9 Working with Swing	255
DAY 10 Building a Swing Interface	285
DAY 11 Arranging Components on a User Interface	313
DAY 12 Responding to User Input	339
DAY 13 Creating Java2D Graphics	367
DAY 14 Developing Swing Applications	391
WEEK III: Java Programming	417
DAY 15 Working with Input and Output	419
DAY 16 Using Inner Classes and Closures	449
DAY 17 Communicating Across the Internet	469
DAY 18 Accessing Databases with JDBC 4.2 and Derby	503
DAY 19 Reading and Writing RSS Feeds	525
DAY 20 XML Web Services	549
DAY 21 Writing Android Apps with Java	569
APPENDIXES	595
A Using the NetBeans Integrated Development Environment	597
B This Book's Website	607
C Fixing a Problem with the Android Studio Emulator	609
D Using the Java Development Kit	615
E Programming with the Java Development Kit	637
Index	661

Table of Contents

- Introduction 1
 - How This Book Is Organized2
 - Who Should Read This Book4
 - Conventions Used in This Book5
- WEEK 1: The Java Language7**
- DAY 1: Getting Started with Java 9**
 - The Java Language10
 - History of the Language10
 - Introduction to Java11
 - Selecting a Development Tool12
 - Object-Oriented Programming13
 - Objects and Classes14
 - Attributes and Behavior17
 - Attributes of a Class of Objects17
 - Behavior of a Class of Objects18
 - Creating a Class19
 - Running the Program22
 - Organizing Classes and Class Behavior25
 - Inheritance25
 - Creating a Class Hierarchy27
 - Inheritance in Action29
 - Interfaces31
 - Packages32
 - Summary32
 - Q&A33
 - Quiz34
 - Questions34
 - Answers35
 - Certification Practice35
 - Exercises35

DAY 2: The ABCs of Programming.	37
Statements and Expressions	38
Variables and Data Types	38
Creating Variables	39
Naming Variables	40
Variable Types	41
Assigning Values to Variables	43
Constants	43
Comments	46
Literals	47
Number Literals	47
Boolean Literals	49
Character Literals	49
String Literals	50
Expressions and Operators	51
Arithmetic	52
More About Assignment	54
Incrementing and Decrementing	55
Comparisons	56
Logical Operators	57
Operator Precedence	58
String Arithmetic	60
Summary	62
Q&A	62
Quiz	63
Questions	63
Answers	64
Certification Practice	64
Exercises	65
DAY 3: Working with Objects	67
Creating New Objects	68
Using new	68
How Objects Are Constructed	70
A Note on Memory Management	71

Using Class and Instance Variables	72
Getting Values	72
Setting Values	73
Class Variables	74
Calling Methods	75
Formatting Strings	77
Nesting Method Calls	78
Class Methods	79
References to Objects	80
Casting Objects and Primitive Types	82
Casting Primitive Types	83
Casting Objects	84
Converting Primitive Types to Objects and Vice Versa	86
Comparing Object Values and Classes	87
Comparing Objects	87
Determining the Class of an Object	89
Summary	90
Q&A	91
Quiz	92
Questions	92
Answers	92
Certification Practice	92
Exercises	93
DAY 4: Lists, Logic, and Loops	95
Arrays	96
Declaring Array Variables	96
Creating Array Objects	97
Accessing Array Elements	98
Changing Array Elements	99
Multidimensional Arrays	102
Block Statements	103
If Conditionals	104
Switch Conditionals	105
The Ternary Operator	112
For Loops	113

While and Do Loops	116
While Loops	116
Do-While Loops	118
Breaking Out of Loops	119
Labeled Loops	120
Summary	121
Q&A	121
Quiz	122
Questions	122
Answers	122
Certification Practice	122
Exercises	123
DAY 5: Creating Classes and Methods	125
Defining Classes	126
Creating Instance and Class Variables	126
Defining Instance Variables	126
Class Variables	127
Creating Methods	127
Defining Methods	128
The <code>this</code> Keyword	130
Variable Scope and Method Definitions	131
Passing Arguments to Methods	132
Class Methods	134
Creating Java Applications	135
Helper Classes	136
Java Applications and Arguments	136
Passing Arguments to Java Applications	137
Handling Arguments in Your Java Application	138
Creating Methods with the Same Name	139
Constructors	144
Basic Constructors	144
Calling Another Constructor	145
Overloading Constructors	146
Overriding Methods	147
Creating Methods That Override Existing Methods	148
Calling the Original Method	149
Overriding Constructors	150

Summary	152
Q&A	153
Quiz	154
Questions	154
Answers	154
Certification Practice	154
Exercises	156

DAY 6: Packages, Interfaces, and Other Class Features..... 157

Modifiers	158
Access Control for Methods and Variables	159
Static Variables and Methods	164
Final Classes, Methods, and Variables	167
Variables	167
Methods	167
Classes	168
Abstract Classes and Methods	169
Packages	169
The <code>import</code> Declaration	171
Class Name Conflicts	172
Creating Your Own Packages	173
Picking a Package Name	173
Creating the Folder Structure	174
Adding a Class to a Package	175
Packages and Class Access Control	175
Interfaces	176
The Problem of Single Inheritance	176
Interfaces and Classes	176
Implementing and Using Interfaces	177
Implementing Multiple Interfaces	177
Other Uses of Interfaces	178
Creating and Extending Interfaces	178
New Interfaces	178
Methods Inside Interfaces	180
Extending Interfaces	180
Creating an Online Storefront	181

Summary	187
Q&A	187
Quiz	188
Questions	188
Answers	188
Certification Practice	189
Exercises	190
DAY 7: Exceptions and Threads	191
Exceptions	192
Exception Classes	194
Managing Exceptions	195
Exception Consistency Checking	195
Protecting Code and Catching Exceptions	196
The <code>finally</code> Clause	199
Declaring Methods That Might Throw Exceptions	202
The <code>throws</code> Clause	203
Which Exceptions Should You Throw?	204
Passing on Exceptions	204
<code>throws</code> and Inheritance	206
Creating and Throwing Exceptions	207
Throwing Exceptions	207
Creating Your Own Exceptions	207
Combining <code>throws</code> , <code>try</code> , and <code>throw</code>	208
When Not to Use Exceptions	210
Bad Style Using Exceptions	210
Threads	211
Writing a Threaded Program	211
A Threaded Application	213
Stopping a Thread	217
Summary	218
Q&A	218
Quiz	219
Questions	219
Answers	220
Certification Practice	220
Exercises	221

WEEK II: The Java Class Library 223**DAY 8: Data Structures 225**

Moving Beyond Arrays.	226
Java Structures	226
Iterator.	228
Bit Sets	229
Array Lists	233
Looping Through Data Structures	235
Stacks	238
Map.	240
Hash Maps	241
Generics.	246
Enumerations.	249
Summary	250
Q&A	251
Quiz.	251
Questions	252
Answers	252
Certification Practice.	252
Exercises	253

DAY 9: Working with Swing 255

Creating an Application	256
Creating an Interface	257
Developing a Framework	260
Creating a Component	261
Adding Components to a Container.	262
Working with Components.	264
Image Icons	265
Labels	267
Text Fields	268
Text Areas	269
Scrolling Panes	271
Check Boxes and Radio Buttons	272
Combo Boxes	274

Lists	276
The Java Class Library	278
Summary	281
Q&A	281
Quiz	281
Questions	282
Answers	282
Certification Practice.	282
Exercises	283
DAY 10: Building a Swing Interface	285
Swing Features	286
Standard Dialog Boxes.	286
Using Dialog Boxes	291
Sliders	294
Scroll Panes	296
Toolbars	297
Progress Bars	300
Menus	303
Tabbed Panes	307
Summary	309
Q&A	310
Quiz	310
Questions	311
Answers	311
Certification Practice.	311
Exercises	312
DAY 11: Arranging Components on a User Interface	313
Basic Interface Layout	314
Laying Out an Interface	314
Flow Layout	315
Box Layout	317
Grid Layout	320
Border Layout.	322
Mixing Layout Managers	324

Card Layout	325
Using Card Layout in an Application	327
Cell Padding and Insets	333
Summary	334
Q&A	334
Quiz	335
Questions	336
Answers	336
Certification Practice	336
Exercises	337
DAY 12: Responding to User Input	339
Event Listeners	340
Setting Up Components	341
Event-Handling Methods	342
Working with Methods	345
Action Events	345
Focus Events	346
Item Events	349
Key Events	351
Mouse Events	352
Mouse Motion Events	352
Window Events	357
Using Adapter Classes	357
Using Inner Classes	359
Summary	362
Q&A	362
Quiz	363
Questions	363
Answers	363
Certification Practice	364
Exercises	365
DAY 13: Creating Java2D Graphics	367
The Graphics2D Class	368
The Graphics Coordinate System	369

Drawing Text	370
Improving Fonts and Graphics with Antialiasing	372
Finding Information About a Font	372
Color	375
Using Color Objects	376
Testing and Setting the Current Colors	376
Drawing Lines and Polygons	377
User and Device Coordinate Spaces	378
Specifying the Rendering Attributes	378
Creating Objects to Draw	381
Drawing Objects	384
Summary	387
Q&A	388
Quiz	388
Questions	388
Answers	389
Certification Practice	389
Exercises	390
DAY 14: Developing Swing Applications	391
Java Web Start	392
Using Java Web Start	395
Creating a JNLP File	396
Supporting Web Start on a Server	405
Additional JNLP Elements	405
Improving Performance with SwingWorker	407
Summary	412
Q&A	413
Quiz	413
Questions	414
Answers	414
Certification Practice	414
Exercises	415

WEEK III: Java Programming 417**DAY 15: Working with Input and Output 419**

Introduction to Streams	420
Using a Stream	420
Filtering a Stream	421
Handling Exceptions	421
Byte Streams	422
File Streams	422
Filtering a Stream	427
Byte Filters	427
Character Streams	437
Reading Text Files	437
Writing Text Files	440
Files and Paths	441
Summary	444
Q&A	444
Quiz	445
Questions	445
Answers	446
Certification Practice	446
Exercises	447

DAY 16: Using Inner Classes and Closures 449

Inner Classes	450
Anonymous Inner Classes	454
Closures	460
Summary	465
Q&A	466
Quiz	466
Questions	467
Answers	467
Certification Practice	467
Exercises	468

DAY 17: Communicating Across the Internet	469
Networking in Java	470
Opening a Stream Over the Net	470
Sockets	475
Socket Servers	479
Testing the Server	482
The <code>java.nio</code> Package	484
Buffers	484
Channels	488
Summary	499
Q&A	499
Quiz	500
Questions	500
Answers	500
Certification Practice	501
Exercises	501
 DAY 18: Accessing Databases with JDBC 4.2 and Derby	 503
Java Database Connectivity	504
Database Drivers	505
Examining a Database	505
Reading Records from a Database	508
Writing Records to a Database	514
Moving Through Resultsets	521
Summary	522
Q&A	523
Quiz	523
Questions	523
Answers	523
Certification Practice	524
Exercises	524
 DAY 19: Reading and Writing RSS Feeds	 525
Using XML	526
Designing an XML Dialect	528
Processing XML with Java	530

Processing XML with XOM	530
Creating an XML Document	532
Modifying an XML Document	536
Formatting an XML Document	540
Evaluating XOM	542
Summary	545
Q&A	546
Quiz	546
Questions	546
Answers	547
Certification Practice	547
Exercises	548
DAY 20: XML Web Services	549
Introduction to XML-RPC	550
Communicating with XML-RPC	551
Sending a Request	551
Responding to a Request	553
Choosing an XML-RPC Implementation	554
Using an XML-RPC Web Service	556
Creating an XML-RPC Web Service	559
Summary	565
Q&A	565
Quiz	566
Questions	566
Answers	566
Certification Practice	566
Exercises	567
DAY 21: Writing Android Apps with Java	569
The History of Android	570
Writing an Android App	571
Organizing an Android Project	574
Creating the Program	575
Running the App	577
Designing an Android App	578
Preparing Resources	579
Configuring a Manifest File	581

Designing the Graphical User Interface	581
Writing Code	584
Summary	592
Q&A	592
Quiz	592
Questions	593
Answers	593
Certification Practice	593
Exercises	594
APPENDIXES	595
APPENDIX A: Using the NetBeans Integrated Development Environment	597
Installing NetBeans	598
Creating a New Project	598
Creating a New Java Class	600
Running the Application	602
Fixing Errors	603
Expanding and Shrinking a Pane	604
Exploring NetBeans	605
APPENDIX B: This Book's Website	607
APPENDIX C: Fixing a Problem with the Android Studio Emulator	609
Problems Running an App	610
Install HAXM in Android Studio	611
Install HAXM on Your Computer	612
Checking BIOS Settings	614
APPENDIX D: Using the Java Development Kit	615
Choosing a Java Development Tool	616
Installing the Java Development Kit	616
Configuring the Java Development Kit	619
Using a Command-Line Interface	619
Opening Folders in MS-DOS	621
Creating Folders in MS-DOS	622
Running Programs in MS-DOS	623
Correcting Configuration Errors	624

Using a Text Editor	628
Creating a Sample Program	629
Compiling and Running the Program in Windows	631
Setting Up the CLASSPATH Variable	632
Setting the Classpath on Most Windows Versions	633
Setting the CLASSPATH on Windows 98 or Me.	635
APPENDIX E: Programming with the Java Development Kit.	637
Overview of the JDK	638
The java Virtual Machine	639
The javac Compiler.	641
The appletviewer Browser	642
The javadoc Documentation Tool	646
The jar Java File Archival Tool.	650
The jdb Debugger	652
Debugging Applications.	653
Debugging Applets.	655
Advanced Debugging Commands	655
Using System Properties.	656
The keytool and jarsigner Code Signing Tools	658
Index.	661

About the Author

Rogers Cadenhead is a programmer and author. He has written more than 30 books on programming and web publishing, including *Sams Teach Yourself Java in 24 Hours* and *Absolute Beginner's Guide to Minecraft Mods Programming*. He also publishes the Drudge Retort and other websites that receive more than 20 million visits a year. He maintains this book's official website at www.java21days.com and a personal weblog at <http://workbench.cadenhead.org>.

Dedication

*To my son Max Cadenhead, who just began his freshman year at art school.
Your talent and dedication to your craft at such an early age makes me proud.*

—Dad

Acknowledgments

A book of this scope (and heft!) requires the hard work and dedication of numerous people. Most of them are at Sams Publishing in Indianapolis, and to them I owe considerable thanks—in particular, to Boris Minkin, Barbara Hacha, Elaine Wiley, and Mark Taber. Most of all, thanks to my wife, Mary, and my sons, Max, Eli, and Sam.

I'd also like to thank readers who have sent helpful comments about corrections, typos, and suggested improvements regarding this book and its prior editions. The list includes the following people: Dave Barton, Patrick Benson, Ian Burton, Lawrence Chang, Jim DeVries, Ryan Esposto, Kim Farr, Sam Fitzpatrick, Bruce Franz, Owen Gailar, Rich Getz, Bob Griesemer, Jenny Guriel, Brenda Henry-Sewell, Ben Hensley, Jon Hereng, Drew Huber, John R. Jackson, Bleu Jaegel, Natalie Kehr, Mark Lehner, Stephen Loscialpo, Brad Kaenel, Chris McGuire, Paul Niedenzu, E.J. O'Brien, Chip Pursell, Pranay Rajgarhia, Peter Riedlberger, Darrell Roberts, Luke Shulenburger, Mike Tomsic, John Walker, Joseph Walsh, Mark Weiss, P.C. Whidden, Chen Yan, Kyu Hwang Yeon, and J-F. Zurcher.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: errata@informit.com

Mail: Addison-Wesley/Prentice Hall Publishing
ATTN: Reader Feedback
330 Hudson Street
7th Floor
New York, New York, 10013

Reader Services

Visit our website and register this book at **informit.com/register** for convenient access to any updates, downloads, or errata that might be available for this book.

Introduction

Some revolutions catch the world by surprise. Twitter, the Linux operating system, and *Pawn Stars* all rose to prominence unexpectedly.

The remarkable success of the Java programming language, on the other hand, caught nobody by surprise. Java has been a source of great expectations since its introduction 20 years ago. When Java was introduced in web browsers, a torrent of publicity welcomed the arrival of the new language.

Sun Microsystems cofounder Bill Joy proclaimed, “This represents the end result of nearly 15 years of trying to come up with a better programming language and environment for building simpler and more reliable software.”

Sun, which created Java in 1991 and first released it to the public four years later, was acquired by Oracle in 2010. Oracle, which has been committed to Java development since its earliest years, has continued to support the language and produce new versions.

In the ensuing years, Java lived up to a considerable amount of its hype. The language has become as strong a part of software development as the beverage of the same name. One kind of Java keeps programmers up nights. The other kind enables programmers to rest easier after they have developed their software.

Java was originally offered as a technology for enhancing websites with programs that run in browsers. Today, it’s more likely to be found on servers, driving dynamic web applications backed by relational databases on some of the web’s largest sites, and on millions of Android cell phones and tablets running popular apps such as Clash of Clans and Instagram.

Each new release of Java strengthens its capabilities as a general-purpose programming language for a wide range of environments. Today, Java is being put to use in desktop applications, Internet servers, mobile devices, and many other environments. It’s even making a comeback in the browser with sophisticated applications created in Java.

Now in its ninth major release—Java 8—the Java language has matured into a full-featured competitor to other general-purpose development languages, such as C++, Python, and Ruby.

You might be familiar with Java programming tools such as Eclipse, NetBeans, and IntelliJ IDEA. These programs make it possible to develop functional Java programs, and you also can use Oracle's Java Development Kit. The kit, which is available for free on the Web, is a set of command-line tools for writing, compiling, and testing Java programs. NetBeans, another free tool offered by Oracle, is an integrated development environment for the creation of Java programs. It can be downloaded from www.netbeans.org.

This book introduces you to all aspects of Java software development using the most current version of the language and the best available techniques in Java Standard Edition, the most widely used version of the language. Programs are prepared and tested using NetBeans, so you can quickly demonstrate the skills you master each day.

Reading this book will help you understand why Java has become the most widely employed programming language on the planet.

How This Book Is Organized

Sams Teach Yourself Java in 21 Days teaches you about the Java language and how to use it to create applications for any computing environment and Android apps that run on cell phones and other mobile devices. By the time you have finished the book, you'll have well-rounded knowledge of Java and the Java class libraries. Using your new skills, you will be able to develop your own programs for tasks such as web services, database connectivity, XML processing, and mobile programming.

You learn by doing in this book, creating several programs each day that demonstrate the topics being introduced. The source code for all these programs is available on the book's official website at www.java21days.com, along with other supplemental material such as answers to reader questions.

This book covers the Java language and its class libraries in 21 days, organized into three weeks. Each week covers a broad area of developing Java programs.

In the first week, you learn about the Java language itself:

- Day 1 covers the basics—what Java is, why you should learn the language, and how to create software using a powerful style of development called object-oriented programming. You create your first Java application.
- On Day 2, you dive into the fundamental Java building blocks—data types, variables, and expressions.

- Day 3 goes into detail about how to deal with objects in Java—how to create them, use their variables, call their methods, and compare them.
- On Day 4, you give Java programs some brainpower using conditionals and work with arrays and loops.
- Day 5 fully explores creating classes—the basic building blocks of any Java program.
- On Day 6, you discover more about interfaces and packages, which are useful for grouping classes and organizing a class hierarchy.
- Day 7 covers three powerful features of Java: exceptions, the ability to deal with errors; threads, the capability to run different parts of a program simultaneously; and assertions, a technique for making programs more reliable.

Week 2 is dedicated to the most useful classes offered by Oracle for use in your own Java programs:

- Day 8 introduces data structures that you can use as an alternative to strings and arrays—array lists, stacks, maps, hash maps, and bit sets. It also describes a special `for` loop that makes them easier to use.
- Day 9 begins a five-day exploration of visual programming. You learn how to create a graphical user interface using Swing classes for interfaces, graphics, and user input.
- Day 10 covers more than a dozen interface components you can use in a Java program, including buttons, text fields, sliders, scrolling text areas, and icons.
- Day 11 explains how to make a user interface look marvelous using *layout managers*, a set of classes that determine how components on an interface are arranged.
- Day 12 concludes the coverage of Swing with event-handling classes, which enable a program to respond to mouse clicks and other user interactions.
- On Day 13, you learn about drawing shapes and characters on user interface components.
- Day 14 demonstrates how to use Java Web Start, a technique that makes installing a Java program as easy as clicking a web page link.

Week 3 moves into advanced topics:

- Day 15 covers input and output using *streams*, a set of classes that enable file access, network access, and other sophisticated data handling.

- Day 16 provides a complete introduction to closures, the most exciting new feature of Java 8. Also called *lambda expressions*, closures make it possible to employ a new type of coding called functional programming in Java for the first time. Inner classes are explored in greater depth as they relate to closures.
- On Day 17, you extend your knowledge of streams to write programs that communicate with the Internet, including socket programming, buffers, channels, and URL handling.
- Day 18 shows you how to connect to relational databases using Java Database Connectivity (JDBC) version 4.2. You learn how to exploit the capabilities of Derby, the open source database that's included with Java.
- Day 19 covers how to read and write RSS documents using the XML Object Model (XOM), an open source Java class library. RSS feeds, one of the most popular XML dialects in use today, enable millions of people to follow site updates and other new web content.
- Day 20 explores how to write web services clients with the language and the Apache XML-RPC class library.
- Day 21 covers the fastest-growing area of Java programming: developing apps for Android phones and mobile devices. Using Google's free Android Studio as a development environment and a free Android development kit, you create apps that can be deployed and tested on a phone.

Who Should Read This Book

This book teaches the Java language to three groups:

- Novices who are relatively new to programming
- People who have been introduced to earlier versions of Java
- Experienced developers in other languages, such as Visual C++, Visual Basic, or Python

When you're finished with this book, you'll be able to tackle any aspect of the Java language. You'll also be comfortable enough to tackle your own ambitious programming projects, both on and off the Web.

If you're somewhat new to programming or have never written a program, you might wonder whether this is the right book for you. Because all the concepts in this book are illustrated with working programs, you'll be able to work your way through the subject regardless of your experience level. If you understand what variables and loops are,

you'll be able to benefit from this book. You might want to read this book if any of the following are true:

- You had some beginning programming lessons in school, you grasp what programming is, and you've heard that Java is easy to learn, powerful, and cool.
- You've programmed in another language for a few years, you keep hearing accolades for Java, and you want to see whether it lives up to its hype.
- You've heard that Java is great for web application and Android programming.

If you've never been introduced to object-oriented programming, which is the style of programming that Java embodies, don't be discouraged. This book assumes that you have no background in object-oriented design. You'll get a chance to learn this development methodology as you're learning Java.

If you're a complete beginner to programming, this book might move a little fast for you. Java is a good language to start with, though, and if you take it slowly and work through all the examples, you can still pick up Java and start creating your own programs.

Conventions Used in This Book

NOTE

A Note presents an interesting, sometimes technical, piece of information related to the discussion.

TIP

A Tip offers advice, such as an easier way to do something.

CAUTION

A Caution advises you of potential problems and helps you steer clear of disaster.

Text that you type and text that appears onscreen is presented in a monospace font:

Monospace looks like this. Hi, mom!

This font represents how text looks onscreen. Placeholders for variables and expressions appear in *monospace italic*.

The end of each lesson offers several special features: answers to commonly asked questions about that day's subject matter, a quiz to test your knowledge of the material, two exercises that you can try on your own, and a practice question in case you're preparing for Java certification. Solutions to the exercises and the answer to the certification question can be found on the book's official website at www.java21days.com.

This page intentionally left blank

DAY 3

Working with Objects

Java is an object-oriented programming language. When you do work in Java, you primarily use objects to get the job done. You create objects, modify them, change their variables, call their methods, and combine them with other objects. You develop classes, create objects out of those classes, and use them with other classes and objects.

Today, you work extensively with objects as you undertake these essential tasks:

- Creating objects
- Testing and modifying their class and instance variables
- Calling an object's methods
- Converting objects from one class to another

Creating New Objects

When you write a Java program, you define a set of classes. As you learned during Day 1, “Getting Started with Java,” a class is a template used to create one or more objects. These objects, which also are called *instances*, are self-contained elements of a program with related features and data. For the most part, you use the class merely to create instances and then work with those instances. In this section, you learn how to create a new object from any given class.

When using strings during Day 2, “The ABCs of Programming,” you learned that a string literal (a series of characters enclosed in double quotation marks) can be used to create a new instance of the class `String` with the value of that string.

The `String` class is unusual in that respect. Although it’s a class, it can be assigned a value with a literal as if it was a primitive data type. This shortcut is available only for strings and classes that represent primitive data types, such as `Integer` and `Double`. To create instances for all other classes, the `new` operator is used.

NOTE

What about the literals for numbers and characters? Don’t they create objects too? Actually, they don’t. The primitive data types for numbers and characters create numbers and characters, but for efficiency they actually aren’t objects. On Day 5, “Creating Classes and Methods,” you learn how to use objects to represent primitive values.

Using `new`

To create a new object, you use the `new` operator with the name of the class that should be used as a template. The name of the class is followed by parentheses, as in these three examples:

```
String name = new String("Hal Jordan");  
URL address = new URL("http://www.java21days.com");  
MarsRobot robbie = new MarsRobot();
```

The parentheses are important and can’t be omitted. They can be empty, however, in which case the most simple, basic object of that class is created. The parentheses also can contain arguments that determine the values of instance variables or other initial qualities of that object.

Here are two objects being created with arguments:

```
Random seed = new Random(606843071);  
Point pt = new Point(0, 0);
```

The number and type of arguments to include inside the parentheses are defined by the class itself using a special method called a *constructor* (which is introduced later today). If you try to create a new instance of a class with the wrong number or wrong type of arguments, or if you give it no arguments and it needs them, an error occurs when the program is compiled.

Today's first project is a demonstration of creating different types of objects with different numbers and types of arguments. The `StringTokenizer` class in the `java.util` package divides a string into a series of shorter strings called *tokens*.

You divide a string into tokens by applying a character or characters as a delimiter. For example, the text "02/20/67" could be divided into three tokens—"02", "20", and "67"—using the slash character / as a delimiter.

Today's first project is a Java application that uses string tokens to analyze stock price data. In NetBeans, create a new empty Java file for the class `TokenTester` in the `com.java21days` package, and enter the code in Listing 3.1 as its source code. This program creates `StringTokenizer` objects by using `new` in two different ways and then displays each token the objects contain.

LISTING 3.1 The Full Text of `TokenTester.java`

```
1: package com.java21days;  
2:  
3: import java.util.StringTokenizer;  
4:  
5: class TokenTester {  
6:  
7:     public static void main(String[] arguments) {  
8:         StringTokenizer st1, st2;  
9:  
10:        String quote1 = "GOOG 530.80 -9.98";  
11:        st1 = new StringTokenizer(quote1);  
12:        System.out.println("Token 1: " + st1.nextToken());  
13:        System.out.println("Token 2: " + st1.nextToken());  
14:        System.out.println("Token 3: " + st1.nextToken());  
15:  
16:        String quote2 = "RHT@75.00@0.22";  
17:        st2 = new StringTokenizer(quote2, "@");  
18:        System.out.println("\nToken 1: " + st2.nextToken());
```

```
19:         System.out.println("Token 2: " + st2.nextToken());
20:         System.out.println("Token 3: " + st2.nextToken());
21:     }
22: }
```

Save this file by choosing File, Save or clicking Save All on the NetBeans toolbar. Run the application by choosing Run, Run File to see the output displayed in Figure 3.1.

FIGURE 3.1

Displaying a `StringTokenizer` object's tokens.



Two different `StringTokenizer` objects are created using different arguments to the constructor.

The first object is created using `new StringTokenizer()` with one argument, a `String` object named `quote1` (line 11). This creates a `StringTokenizer` object that uses the default delimiters, which are blank spaces, tabs, newlines, carriage returns, or formfeed characters.

If any of these characters is contained in the string, it is used to divide the string. Because the `quote1` string contains spaces, these are used as delimiters dividing each token. Lines 12–14 display the values of all three tokens: “GOOG”, “530.80”, and “-9.98”.

The second `StringTokenizer` object in this example has two arguments when it is constructed in line 16—a `String` object named `quote2` and an at-sign character `@`. This second argument indicates that the `@` character should be used as the delimiter between tokens. The `StringTokenizer` object created in line 17 contains three tokens: “RHT”, “75.00”, and “0.22”.

How Objects Are Constructed

Several things happen when you use the `new` operator. The new instance of the given class is created, memory is allocated for it, and a special method defined in the given class is called. This method is called a constructor.

A *constructor* is a way to create a new instance of a class. A constructor initializes the new object and its variables, creates any other objects that the object needs, and performs any additional operations the object requires to initialize itself.

A class can have several different constructors, each with a different number or type of argument. When you use `new`, you can specify different arguments in the argument list, and the correct constructor for those arguments is called.

In the `TokenTester` project, multiple constructor definitions enabled the `StringTokenizer` class to accomplish different things with different uses of the `new` operator. When you create your own classes, you can define as many constructors as you need to implement the behavior of the class.

No two constructors in a class can have the same number and type of arguments, because this is the only way constructors are differentiated from each other.

If a class defines no constructors, a constructor with no arguments is called by default when an object of the class is created. The only thing this constructor does is call the same constructor in its superclass.

CAUTION

The default constructor only exists in a class that has not defined any constructors. Once you define at least one constructor in a class, you can't count on there being a default constructor with no arguments.

A Note on Memory Management

If you are familiar with other object-oriented programming languages, you might wonder whether the `new` operator has an opposite that destroys an object when it is no longer needed.

Memory management in Java is dynamic and automatic. When you create a new object, Java automatically allocates the proper amount of memory for that object. You don't have to allocate any memory for objects explicitly. The Java Virtual Machine (JVM) does it for you.

Because Java memory management is automatic, you don't need to deallocate the memory an object uses when you're finished using it. Under most circumstances, when you are finished with an object you have created, Java can determine that the object no longer has any live references to it. (In other words, the object isn't assigned to any variables still in use or stored in any arrays.)

As a program runs, the JVM periodically looks for unused objects and reclaims the memory that those objects are using. This process is called *dynamic garbage collection* and occurs without any programming on your part. You don't have to explicitly free the memory taken up by an object; you just have to make sure that you're not still holding onto an object you want to get rid of.

This feature is one of the most touted advantages of the language over its predecessor C++.

Using Class and Instance Variables

At this point, you can create your own object with class and instance variables, but how do you work with those variables? They're used in largely the same manner as the local variables you learned about yesterday. You can put them in expressions, assign values to them in statements, and so on. You just refer to them slightly differently.

Getting Values

To get to the value of an instance variable, you use *dot notation*, a form of addressing in which an instance or class variable name has two parts:

- A reference to an object or class on the left side of a dot operator .
- A variable on the right side

Dot notation is how you refer to an object's instance variables and methods.

For example, if you have an object named `customer` with a variable called `orderTotal`, here's how that variable could be referred to in a statement:

```
float total = customer.orderTotal;
```

This statement assigns the value of the `customer` object's `orderTotal` instance variable to a floating-point variable named `total`.

Accessing variables in dot notation is an expression (meaning that it returns a value). Both sides of the dot also are expressions. This means that you can chain instance variable access.

Extending the preceding example, suppose the `customer` object is an instance variable of the `store` class. Dot notation can be used twice, as in this statement:

```
float total = store.customer.orderTotal;
```

Dot expressions are evaluated from left to right, so you start with `store`'s instance variable `customer`, which itself has an instance variable `orderTotal`. The value of this variable is assigned to the `total` variable.

One thing to note when chaining objects together in this manner is that the statement will fail with an error if any object in the chain does not have a value yet.

Setting Values

Assigning a value to an instance variable with dot notation employs the `=` operator just like variables holding primitive types:

```
customer.layaway = true;
```

This example sets the value of a `boolean` instance variable named `layaway` to `true`.

The `PointSetter` application in Listing 3.2 tests and modifies the instance variables in a `Point` object. `Point`, a class in the `java.awt` package, represents points in a coordinate system with (x, y) values.

Create a new empty Java file in NetBeans with the class name `PointSetter` and the package name `com.java21days`; then type the source code shown in Listing 3.2 and save the file.

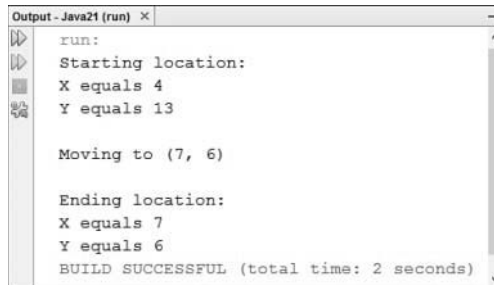
LISTING 3.2 The Full Text of `PointSetter.java`

```
1: package com.java21days;
2:
3: import java.awt.Point;
4:
5: class PointSetter {
6:
7:     public static void main(String[] arguments) {
8:         Point location = new Point(4, 13);
9:
10:        System.out.println("Starting location:");
11:        System.out.println("X equals " + location.x);
12:        System.out.println("Y equals " + location.y);
13:
14:        System.out.println("\nMoving to (7, 6)");
15:        location.x = 7;
16:        location.y = 6;
17:
18:        System.out.println("\nEnding location:");
19:        System.out.println("X equals " + location.x);
20:        System.out.println("Y equals " + location.y);
21:    }
22: }
```

When you run this application, the output should match Figure 3.2.

FIGURE 3.2

Setting and displaying an object's instance variables.



```
Output - Java21 (run) X
run:
Starting location:
X equals 4
Y equals 13

Moving to (7, 6)

Ending location:
X equals 7
Y equals 6
BUILD SUCCESSFUL (total time: 2 seconds)
```

In this application, you create an instance of `Point` where `x` equals 4 and `y` equals 13 (line 8). These individual values are retrieved using dot notation.

The value of `x` is changed to 7 and `y` to 6 (lines 15–16). The values are displayed again to show how they have changed.

Class Variables

Class variables, as you have learned, are variables defined and stored in the class itself. Their values apply to the class and all its instances.

With instance variables, each new instance of the class gets a new copy of the instance variables that the class defines. Each instance then can change the values of those instance variables without affecting any other instances. With class variables, only one copy of that variable exists when the class is loaded. Changing the value of that variable changes it for all instances of that class.

You define class variables by including the `static` keyword before the variable itself. For example, consider the following partial class definition:

```
class FamilyMember {
    static String surname = "Mendoza";
    String name;
    int age;
}
```

Each instance of the class `FamilyMember` has its own values for `name` and `age`, but the class variable `surname` has only one value for all family members: “Mendoza.” If the value of `surname` is changed, all instances of `FamilyMember` are affected.

NOTE

Calling these static variables refers to one of the meanings of the word “static”: fixed in one place. If a class has a `static` variable, every object of that class has the same value for that variable.

To access class variables, you use the same dot notation as with instance variables. To retrieve or change the value of the class variable, you can use either the instance or the name of the class on the left side of the dot operator. Both lines of output in this example display the same value:

```
FamilyMember dad = new FamilyMember();
System.out.println("Family's surname is: " + dad.surname);
System.out.println("Family's surname is: " + FamilyMember.surname);
```

Because you can use an object to change the value of a class variable, it’s easy to become confused about class variables and where their values are coming from. Remember that the value of a class variable affects all objects of that particular class. If the `surname` instance variable of one `FamilyMember` object was set to “Paciorek”, all objects of that class would have that new surname.

To reduce confusion when using class variables, it’s a good idea to use the name of the class when you refer to a class variable—not an object of that class. This makes the use of a class variable more clear and helps strange results become easier to debug.

Calling Methods

Methods of an object are called to make it do something.

Calling a method in an object also makes use of dot notation. The object whose method is being called is on the left side of the dot, and the name of the method and its arguments are on the right side:

```
customer.addToCart(itemNumber, price, quantity);
```

All method calls must have parentheses after them, even when the method takes no arguments, as in this example:

```
customer.cancelOrder();
```

In Listing 3.3, the `StringChecker` application shows an example of calling some methods defined in the `String` class. Strings include methods for string tests and modification. Create this program in NetBeans as an empty Java file with the class name `StringChecker` and package name `com.java21days`.

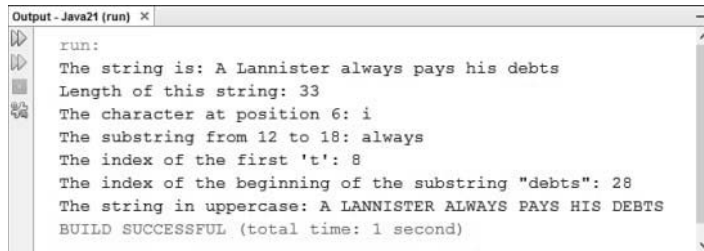
LISTING 3.3 The Full Text of StringChecker.java

```
1: package com.java21days;
2:
3: class StringChecker {
4:
5:     public static void main(String[] arguments) {
6:         String str = "A Lannister always pays his debts";
7:         System.out.println("The string is: " + str);
8:         System.out.println("Length of this string: "
9:             + str.length());
10:        System.out.println("The character at position 6: "
11:            + str.charAt(6));
12:        System.out.println("The substring from 12 to 18: "
13:            + str.substring(12, 18));
14:        System.out.println("The index of the first 't': "
15:            + str.indexOf('t'));
16:        System.out.println("The index of the beginning of the "
17:            + "substring \"debts\": " + str.indexOf("debts"));
18:        System.out.println("The string in uppercase: "
19:            + str.toUpperCase());
20:    }
21: }
```

Running the program produces the output shown in Figure 3.3.

FIGURE 3.3

Calling String methods to learn more about that string.



```
run:
The string is: A Lannister always pays his debts
Length of this string: 33
The character at position 6: i
The substring from 12 to 18: always
The index of the first 't': 8
The index of the beginning of the substring "debts": 28
The string in uppercase: A LANNISTER ALWAYS PAYS HIS DEBTS
BUILD SUCCESSFUL (total time: 1 second)
```

In line 6, you create a new instance of `String` by using the string literal “A Lannister always pays his debts”. The remainder of the program simply calls different string methods to do different operations on that string:

- Line 7 prints the value of the string.
- Line 9 calls the `length()` method in the new `String` object to find out how many characters it contains.

- Line 11 calls the `charAt()` method, which returns the character at the given position in the string. Note that string positions start at position 0 rather than 1, so the character at position 6 is 'i'.
- Line 13 calls the `substring()` method, which takes two integers indicating a range and returns the substring with those starting and ending points. The `substring()` method also can be called with only one argument, which returns the substring from that position to the end of the string.
- Line 15 calls the `indexOf()` method, which returns the position of the first instance of the given character. Character literals are surrounded by single quotation marks, so the argument is 't' (not "t").
- Line 17 shows a different use of the `indexOf()` method, which takes a string argument and returns the index of the beginning of that string. String literals always are surrounded by double quotation marks.
- Line 19 uses the `toUpperCase()` method to return a copy of the string in all uppercase.

NOTE

If you compare the output of the `StringChecker` application to the characters in the string, you might be wondering how 'i' could be at position 6 when it is the seventh character in the string. All of the methods look like they're off by one (except for `length()`). The reason is that the methods are zero-based, which means they begin counting with 0 instead of 1. So 'A' is at position 0, a space at position 1, 'L' at position 2 and so on. This kind of numbering is something you encounter often in Java.

Formatting Strings

Numbers such as money often need to be displayed in a precise manner. There are only two places after the decimal for the number of cents, a dollar sign (\$) preceding the value, and commas separating groups of three numbers—as in \$22,453.70 (the amount the U.S. National Debt goes up in one second).

This kind of formatting when displaying strings can be accomplished with the `System.out.format()` method.

The method takes two arguments: the output format template and the string to display. Here's an example that adds a dollar sign and commas to the display of an integer:

```
int accountBalance = 5005;  
System.out.format("Balance: $%,d%n", accountBalance);
```

This code produces the output `Balance: $5,005`.

The formatting string begins with a percent sign `%` followed by one or more flags. The `%,d` code displays a decimal with commas dividing each group of three digits. The `%n` code displays a newline character.

The next example displays the value of pi to 11 decimal places:

```
double pi = Math.PI;  
System.out.format("%.11f%n", pi);
```

The output is `3.14159265359`.

TIP

Oracle's Java site includes a beginner's tutorial for `printf`-style output that describes some of the most useful formatting codes: <http://docs.oracle.com/javase/tutorial/java/data/numberformat.html>

Nesting Method Calls

A method can return a reference to an object, a primitive data type, or no value at all. In the `StringChecker` application, all the methods called on the `String` object `str` returned values that are displayed. The `charAt()` method returned a character at a specified position in the string.

The value returned by a method also can be stored in a variable:

```
String label = "From";  
String upper = label.toUpperCase();
```

In this example, the `String` object `upper` contains the value returned by calling `label.toUpperCase()`, which is the text `"FROM"`.

If the method returns an object, you can call the methods of that object in the same statement. This makes it possible for you to nest methods as you would variables.

Earlier today, you saw an example of a method called with no arguments:

```
customer.cancelOrder();
```

If the `cancelOrder()` method returns an object, you can call methods of that object in the same statement:

```
customer.cancelOrder().fileComplaint();
```

This statement calls the `fileComplaint()` method, which is defined in the object returned by the `cancelOrder()` method of the `customer` object.

You can combine nested method calls and instance variable references as well. In the next example, the `putOnLayaway()` method is defined in the object stored by the `orderTotal` instance variable, which itself is part of the `customer` object:

```
customer.orderTotal.putOnLayaway(itemNumber, price, quantity);
```

This manner of nesting variables and methods is demonstrated in a method you've used frequently in the first three days of this book: `System.out.println()`.

That method displays strings and other data to the computer's standard output device.

The `System` class, part of the `java.lang` package, describes behavior specific to the computer system on which Java is running. `System.out` is a class variable that contains an instance of the class `PrintStream` representing the system's standard output, which normally is the monitor but can be a printer or file. `PrintStream` objects have a `println()` method that sends a string to that output stream. The `PrintStream` class is in the `java.io` package.

Class Methods

Class methods, also called static methods, apply to the class as a whole and not to its instances just like class variables. Class methods commonly are used for general utility methods that might not operate directly on an object of that class but do fit with that class conceptually.

For example, the `String` class contains a class method called `valueOf()`, which can take one of many types of arguments (integers, Booleans, objects, and so on). The `valueOf()` method then returns a new instance of `String` containing the argument's string value. This method doesn't operate directly on an existing instance of `String`, but getting a string from another object or data type is behavior that makes sense to define in the `String` class.

Class methods also can be useful for gathering general methods in one place. For example, the `Math` class, defined in the `java.lang` package, contains a large set of

mathematical operations as class methods. No objects can be created from the `Math` class, but you still can use its methods with numeric or Boolean arguments.

For example, the class method `Math.max()` takes two arguments and returns the larger of the two. You don't need to create a new instance of `Math`; it can be called anywhere you need it, as in the following:

```
int firstPrice = 225;
int secondPrice = 217;
int higherPrice = Math.max(firstPrice, secondPrice);
```

Dot notation is used to call a class method. As with class variables, you can use either an instance of the class or the class itself on the left side of the dot. For the same reasons noted earlier about class variables, using the name of the class makes your code easier to read.

The last two lines in this example both produce strings equal to “550”:

```
String s, s2;
s = "potrzebie";
s2 = s.valueOf(550);
s2 = String.valueOf(550);
```

References to Objects

As you work with objects, it's important to understand references. A *reference* is an address that indicates where an object's variables and methods are stored.

You aren't actually using objects when you assign an object to a variable or pass an object to a method as an argument. You aren't even using copies of the objects. Instead, you're using references to those objects.

To better illustrate what this means, the `RefTester` application in Listing 3.4 shows how references work. Create an empty Java file in NetBeans for the class `RefTester` in the package `com.java21days`, and enter Listing 3.4 as the application's source code.

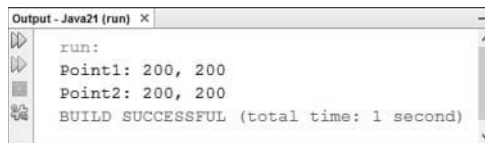
LISTING 3.4 The Full Text of `RefTester.java`

```
1: package com.java21days;
2:
3: import java.awt.Point;
4:
5: class RefTester {
6:     public static void main(String[] arguments) {
7:         Point pt1, pt2;
```

```
8:         pt1 = new Point(100, 100);
9:         pt2 = pt1;
10:
11:        pt1.x = 200;
12:        pt1.y = 200;
13:        System.out.println("Point1: " + pt1.x + ", " + pt1.y);
14:        System.out.println("Point2: " + pt2.x + ", " + pt2.y);
15:    }
16: }
```

Save and run the application. The output is shown in Figure 3.4.

FIGURE 3.4
Putting references
to a test.



The following takes place in the first part of this program:

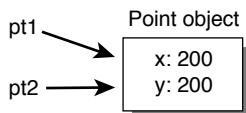
- **Line 7**—Two `Point` variables are created.
- **Line 8**—A new `Point` object is assigned to `pt1`.
- **Line 9**—The variable `pt1` is assigned to `pt2`.

Lines 11–14 are the tricky part. The `x` and `y` variables of `pt1` both are set to 200 and all variables of `pt1` and `pt2` are displayed onscreen.

You might expect `pt1` and `pt2` to have different values, but Figure 3.4 shows this not to be the case. The `x` and `y` variables of `pt2` also have changed even though nothing in the program explicitly changes them. This happens because line 7 creates a reference from `pt2` to `pt1` instead of creating `pt2` as a new object copied from `pt1`.

The variable `pt2` is a reference to the same object as `pt1`, as shown in Figure 3.5. Either variable can be used to refer to the object or to change its variables.

FIGURE 3.5
References to
objects.



If you wanted `pt1` and `pt2` to refer to separate objects, you could use separate `new Point()` statements on lines 6–7 to create separate objects, as shown here:

```
pt1 = new Point(100, 100);  
pt2 = new Point(100, 100);
```

References in Java become particularly important when arguments are passed to methods. You learn more about this later today.

NOTE

Java has no explicit pointers or pointer arithmetic, unlike C and C++. By using references and Java arrays, you can duplicate most pointer capabilities without many of their drawbacks.

Casting Objects and Primitive Types

One thing you discover quickly about Java is how finicky it is about the information it will handle. Like Goldilocks, the child who is oddly hard to please about porridge for a person who breaks into homes, Java methods and constructors require things to take a specific form and won't accept alternatives.

When you send arguments to methods or use variables in expressions, you must use variables of the correct data types. If a method requires an `int`, the Java compiler responds with an error if you try to send a `float` value to the method. Likewise, if you set up one variable with the value of another, they must be of compatible types. The two variables must be the same type or the variable receiving the value must be big enough to hold the value.

NOTE

There is one area where Java's compiler is decidedly flexible: the `String` object. String handling in `println()` methods, assignment statements, and method arguments is simplified by the `+` concatenation operator. If any variable in a group of concatenated variables is a string, Java treats the whole thing as a `String`. This makes the following possible:

```
float gpa = 2.25F;  
System.out.println("Honest, mom, my GPA is a " +  
    (gpa + 1.5));
```

Using the concatenation operator, a single string can hold the text representation of multiple objects and primitive types in Java.

Sometimes you have a value in your Java class that isn't the right type for what you need. It might be the wrong class or the wrong data type, such as a `float` when you need an `int`.

In these situations, you can use a process called *casting* to convert a value from one type to another.

Although casting is reasonably simple, the process is complicated by the fact that Java has both primitive types (such as `int`, `float`, and `boolean`) and object types (`String`, `Point`, and the like). This section discusses three forms of casts and conversions:

- Casting between primitive types, such as `int` to `float` or `float` to `double`
- Casting from an object of a class to an object of another class, such as from `Object` to `String`
- Casting primitive types to objects and then extracting primitive values from those objects

When discussing casting, it can be easier to think in terms of sources and destinations. The source is the variable being cast into another type. The destination is the result.

Casting Primitive Types

Casting between primitive types enables you to convert the value of one type to another. This most commonly occurs with the numeric types, but one primitive type never can be used in a cast. Boolean values must be either `true` or `false` and cannot be used in a casting operation.

In many casts between primitive types, the destination can hold larger values than the source, so the value is converted easily. An example would be casting a `byte` into an `int`. Because a `byte` holds values from `-128` to `127` and an `int` holds from around `-2,100,000` to `2,100,000`, there's more than enough room to cast a `byte` into an `int`.

Often you can automatically use a `byte` or `char` as an `int`; an `int` as a `long`; an `int` as a `float`; or anything as a `double`. In most cases, because the larger type provides more precision than the smaller, no loss of information occurs as a result. The exception is casting integers to floating-point values. Casting a `long` to a `float` or a `long` to a `double` can cause some loss of precision.

NOTE

A character can be used as an `int` because each character has a corresponding numeric code that represents its position in the character set. If the variable `key` has the value 65, the cast `(char) key` produces the character value `'A'`. The numeric code associated with a capital A is 65 in the ASCII character set, which Java adopted as part of its character support.

You must use an explicit cast to convert a value in a large type to a smaller type. Explicit casts take the following form:

```
(typename) value
```

Here *typename* is the name of the primitive data type to which you're converting, such as `short`, `int`, or `float`. *value* is an expression that results in the value of the source type. For example, in the following statement, the value of `x` is divided by the value of `y` and the result is cast into an `int`:

```
int result = (int)(x / y);
```

Note that because the precedence of casting is higher than that of arithmetic, you have to use parentheses here. Otherwise, first the value of `x` would be cast into an `int`, and then it would be divided by `y`, which could produce a different result.

Casting Objects

Objects of classes also can be cast into objects of other classes when the source and destination classes are related by inheritance and one class is a subclass of the other.

Some objects might not need to be cast explicitly. In particular, because a subclass contains all the information as its superclass, you can use an object of a subclass anywhere a superclass is expected.

For example, consider a method that takes two arguments, one of type `Object` and another of type `Component` in the `java.awt` package (which has classes for a graphical user interface).

You can pass an instance of any class for the `Object` argument because all Java classes are subclasses of `Object`.

For the `Component` argument, you can pass in its subclasses, such as `Button`, `Container`, and `Label` (all in `java.awt`).

This is true anywhere in a program, not just inside method calls. If you had a variable defined as class `Component`, you could assign objects of that class or any of its subclasses to that variable without casting.

This also is true in the reverse, so you can use a superclass when a subclass is expected. There is a catch, however: Because subclasses contain more behavior than their superclasses, a loss of precision occurs in the casting. Those superclass objects might not have all the behavior needed to act in place of a subclass object.

Consider this example: If you have an operation that calls methods in objects of the class `Integer`, using an object of its superclass `Number` won't include many methods specified in `Integer`. Errors occur if you try to call methods that the destination object doesn't have.

To use superclass objects where subclass objects are expected, you must cast them explicitly. You won't lose any information in the cast, but you gain all the methods and variables that the subclass defines.

To cast an object to another class, you use the same operation as for primitive types, which takes this form:

```
(classname) object
```

In this template, *classname* is the name of the destination class, and *object* is a reference to the source object. Casting creates a reference to the old object of the type *classname*; the old object continues to exist as it did before.

The following example casts an instance of the class `VicePresident` to an instance of the class `Employee`. `VicePresident` is a subclass of `Employee` with more information:

```
Employee emp = new Employee();  
VicePresident veep = new VicePresident();  
emp = veep; // no cast needed for upward use  
veep = (VicePresident) emp; // must cast explicitly
```

When you begin working with graphical user interfaces during Week 2, “The Java Class Library,” you will see that casting one object is necessary whenever you use Java2D graphics operations. You must cast a `Graphics` object to a `Graphics2D` object before you can draw onscreen. The following example uses a `Graphics` object called `screen` to create a new `Graphics2D` object called `screen2D`:

```
Graphics2D screen2D = (Graphics2D) screen;
```

`Graphics2D` is a subclass of `Graphics` and both belong to the `java.awt` package. You'll explore this subject fully during Day 13, “Creating Java2D Graphics.”

In addition to casting objects to classes, you can cast objects to interfaces, but only if an object's class or one of its superclasses actually implements the interface. Casting an object to an interface means that you can call one of that interface's methods even if that object's class does not actually implement that interface.

Converting Primitive Types to Objects and Vice Versa

One thing you can't do is cast from an object to a primitive data type, or vice versa.

Primitive types and objects are different things in Java, and you can't automatically cast between the two.

As an alternative, the `java.lang` package includes classes that correspond to each primitive data type: `Float`, `Boolean`, `Byte`, and so on. Most of these classes have the same names as the data types, except that the class names begin with a capital letter (`Short` instead of `short`, `Double` instead of `double`, and the like). Also, two classes have names that differ from the corresponding data type: `Character` is used for `char` variables and `Integer` is used for `int` variables.

Using the classes that correspond to each primitive type, you can create an object that holds the same value. The following statement creates an instance of the `Integer` class with the integer value 7801:

```
Integer dataCount = new Integer(7801);
```

After you have created an object in this manner, you can use it as you would any object (although you cannot change its value). When you want to use that value again as a primitive value, there are methods for that as well. For example, if you wanted to get an `int` value from a `dataCount` object, the following statement shows how:

```
int newCount = dataCount.intValue(); // returns 7801
```

A common translation you need in programs is converting a `String` to a numeric type, such as an integer. When you need an `int` as the result, this can be done by using the `parseInt()` class method of the `Integer` class. The `String` to convert is the only argument sent to the method, as in the following example:

```
String pennsylvania = "65000";  
int penn = Integer.parseInt(pennsylvania);
```

The following classes can be used to work with objects instead of primitive data types: `Boolean`, `Byte`, `Character`, `Double`, `Float`, `Integer`, `Long`, `Short`, and `Void`. These classes are commonly called *object wrappers* because they provide an object representation that contains a primitive value.

CAUTION

If you try to use the preceding example in a program, your program won't compile. The `parseInt()` method is designed to fail with a `NumberFormatException` error if the argument to the method is not a valid numeric value. To deal with errors of this kind, you must use special error-handling statements, which are introduced during Day 7, "Exceptions and Threads."

Working with primitive types and objects that represent the same values is made easier through autoboxing and unboxing, an automatic conversion process.

Autoboxing automatically converts a primitive type to an object. *Unboxing* converts in the other direction.

If you write a statement that uses an object where a primitive type is expected, or vice versa, the value is converted so that the statement executes successfully.

Here's an example of autoboxing and unboxing:

```
Float f1 = 12.5F;  
Float f2 = 27.2F;  
System.out.println("Lower number: " + Math.min(f1, f2));
```

The `Math.min()` method takes two `float` values as arguments, but the preceding example sends the method two `Float` objects as arguments instead.

The compiler does not report an error over this discrepancy. Instead, the `Float` objects automatically are unboxed into `float` values before being sent to the `min()` method.

CAUTION

Unboxing an object works only if the object has a value. If no constructor has been called to set up the object, compilation fails with an error.

Comparing Object Values and Classes

In addition to casting, you often will perform three other common tasks that involve objects:

- Comparing objects
- Finding out the class of any given object
- Testing whether an object is an instance of a given class

Comparing Objects

Yesterday, you learned about operators for comparing values—equal, not equal, less than, and so on. Most of these operators work only on primitive types, not on objects. If you try to use other values as operands, the Java compiler produces errors.

The exceptions to this rule are the `==` operator for equality and the `!=` operator for inequality. When applied to objects, these operators don't do what you might first expect. Instead of checking whether one object has the same value as the other, they determine whether both sides of the operator refer to the same object.

To compare objects of a class and have meaningful results, you must implement special methods in your class and call those methods.

A good example of this is the `String` class. It is possible to have two different `String` objects that represent the same text. If you were to employ the `==` operator to compare these objects, however, they would be considered unequal. Although their contents match, they are not the same object.

To see whether two `String` objects have matching values, an `equals()` method of the class is used. The method tests each character in the string and returns `true` if the two strings have the same value. The `EqualsTester` application shown in Listing 3.5 illustrates this. Create the application with NetBeans in the `com.java21days` package and save the file, either by choosing File, Save or clicking the Save All toolbar button.

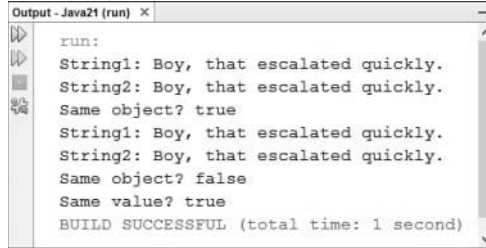
LISTING 3.5 The Full Text of `EqualsTester.java`

```
1: package com.java21days;
2:
3: class EqualsTester {
4:     public static void main(String[] arguments) {
5:         String str1, str2;
6:         str1 = "Boy, that escalated quickly.";
7:         str2 = str1;
8:
9:         System.out.println("String1: " + str1);
10:        System.out.println("String2: " + str2);
11:        System.out.println("Same object? " + (str1 == str2));
12:
13:        str2 = new String(str1);
14:
15:        System.out.println("String1: " + str1);
16:        System.out.println("String2: " + str2);
17:        System.out.println("Same object? " + (str1 == str2));
18:        System.out.println("Same value? " + str1.equals(str2));
19:    }
20: }
```

The program's output appears in Figure 3.6.

FIGURE 3.6

Calling `String` methods to learn more about that string.



```
Output - Java21 (run) x
run:
String1: Boy, that escalated quickly.
String2: Boy, that escalated quickly.
Same object? true
String1: Boy, that escalated quickly.
String2: Boy, that escalated quickly.
Same object? false
Same value? true
BUILD SUCCESSFUL (total time: 1 second)
```

The first part of this program declares two variables (`str1` and `str2`), assigns the literal “Boy, that escalated quickly.” to `str1`, and then assigns that value to `str2` (lines 5–7). As you learned earlier, `str1` and `str2` now point to the same object, and the equality test at line 11 proves that.

In the second part of this program, you create a new `String` object with the same value as `str1` and assign `str2` to that new `String` object.

Now you have two different string objects in `str1` and `str2`, both with the same value. Testing them to see whether they're the same object by using the `==` operator returns the expected answer: `false` (line 17). They are not the same object in memory. Testing them using the `equals()` method in line 18 also returns the expected answer of `true`, which shows they have the same value.

NOTE

Why can't you just use another literal when you change `str2`, instead of using `new`? String literals are optimized in Java. If you create a string using a literal and then use another literal with the same characters, Java gives you back the first `String` object. Both strings are the same object; you have to go out of your way to create two separate objects.

Determining the Class of an Object

Want to find out the name of an object's class? Here's how for an object assigned to the variable `key`:

```
String name = key.getClass().getName();
```

The `getClass()` method is defined in the `Object` class, so it can be called in all Java objects. The method returns a `Class` object that represents the object's class. That object's `getName()` method returns a string holding the name of the class.

Another useful test is the `instanceof` operator, which has two operands: a reference to an object on the left, and a class name on the right. The expression produces a Boolean value: `true` if the object is an instance of the named class or any of that class's subclasses, or `false` otherwise, as in these examples:

```
boolean check1 = "Texas" instanceof String; // true

Object obiwan = new Object();
boolean check2 = obiwan instanceof String; // false
```

The `instanceof` operator also can be used for interfaces. If an object implements an interface, the `instanceof` operator returns `true` when this is tested.

Unlike other operators in Java, `instanceof` is not a form of punctuation like `*` for multiplication or `+` for addition. Instead, the `instanceof` keyword is the operator.

Summary

Now that you have spent three days exploring how object-oriented programming is implemented in Java, you're in a better position to decide how useful it can be in your programming.

If you are a “glass half empty” kind of person, object-oriented programming (OOP) is a level of abstraction that gets in the way of using a programming language. You learn more about why OOP is thoroughly ingrained in Java in the coming days and may change your mind.

If you are a “glass half full” kind of person, object-oriented programming is beneficial because of its benefits: improved reliability, reusability, and maintenance.

Today you learned how to deal with objects: creating them, reading their values and changing them, and calling their methods. You also learned how to cast objects from one class to another, cast to and from primitive data types and classes, and take advantage of automatic conversions through autoboxing and unboxing.

Q&A

Q I'm confused about the differences between objects and the primitive data types, such as `int` and `boolean`.

A The primitive types (`byte`, `short`, `int`, `long`, `float`, `double`, `boolean`, and `char`) are not objects, although in many ways they can be handled like objects. They can be assigned to variables and passed in and out of methods.

Objects are instances of classes and as such are much more complex data types than simple numbers and characters. They often contain numbers and characters as instance or class variables.

Q The `length()` and `charAt()` methods in the `StringChecker` application (Listing 3.3) don't appear to make sense. If `length()` says that a string is 33 characters long, shouldn't the characters be numbered from 1 to 33 when `charAt()` is used to display characters in the string?

A The two methods look at strings differently. The `length()` method counts the characters in the string, with the first character counting as 1, the second as 2, and so on. The `charAt()` method considers the first character in the string to be located at position number 0. This is the same numbering system used with array elements in Java. Consider the string "Charlie Brown". It has 13 characters ranging from position 0 (the letter C) to position 12 (the letter n).

Q If Java lacks pointers, how can I do something like linked lists, where there's a pointer from one node to another so that they can be traversed?

A It's incorrect to say that Java has no pointers; it just has no *explicit* pointers. Object references are effectively pointers. To create something like a linked list, you could create a class called `Node`, which would have an instance variable also of type `Node`. To link node objects, assign a node object to the instance variable of the object immediately before it in the list. Because object references are pointers, linked lists set up this way behave as you would expect them to. (You'll work with the Java class library's version of linked lists on Day 8, "Data Structures.")

Quiz

Review today's material by taking this three-question quiz.

Questions

1. What operator do you use to call an object's constructor and create a new object?
 - A. `+`
 - B. `new`
 - C. `instanceof`
2. What kind of methods apply to all objects of a class rather than an individual object?
 - A. Universal methods
 - B. Instance methods
 - C. Class methods
3. If you have a program with objects named `obj1` and `obj2`, what happens when you use the statement `obj2 = obj1`?
 - A. The instance variables in `obj2` are given the same values as `obj1`.
 - B. `obj2` and `obj1` are considered to be the same object.
 - C. Neither A nor B.

Answers

1. B. The `new` operator is followed by a call to the object's constructor.
2. C. Class methods can be called without creating an object of that class.
3. B. The `=` operator does not copy values from one object to another. Instead, it makes both variables refer to the same object.

Certification Practice

The following question is the kind of thing you could expect to be asked on a Java programming certification test. Answer it without looking at today's material or using the Java compiler to test the code.

Given:

```
public class AyeAye {  
    int i = 40;  
    int j;
```

```
public AyeAye() {  
    setValue(i++);  
}  
  
void setValue(int inputValue) {  
    int i = 20;  
    j = i + 1;  
    System.out.println("j = " + j);  
}  
}
```

What is the value of the `j` variable at the time it is displayed inside the `setValue()` method?

- A. 42
- B. 40
- C. 21
- D. 20

The answer is available on the book's website at www.java21days.com. Visit the Day 3 page and click the Certification Practice link.

Exercises

To extend your knowledge of the subjects covered today, try the following exercises:

1. Create a program that turns a birthday in MM/DD/YYYY format (such as 04/29/2016) into three individual strings.
2. Create a class with instance variables for `height`, `weight`, and `depth`, making each an integer. Create a Java application that uses your new class, sets each of these values in an object, and displays the values.

Exercise solutions are offered on the book's website at www.java21days.com.

This page intentionally left blank

This page intentionally left blank

Index

Symbols

- \$ (dollar sign) in variable names, 41
- & (ampersand), AND operators, 57
- > (arrow operator), 460
- * (asterisk) multiplication operator, 52
- *= (asterisk equal) assignment operator, 54
- ^ (caret) XOR operator, 58
- !! command (jdb), 654
- /* comment notation, 46
- /** comment notation, 47
- // comment notation, 46
- { (curly braces), 38
 - block statements, 103
- <> (diamond operator), 247
- && (double ampersand), AND operators, 57
- || (double pipe character), OR operators, 58
- == (equal) comparison operator, 57, 88
- = (equal sign) assignment operator, 40, 43, 54-55
- ! (exclamation point) NOT operator, 58
- / (forward slash) division operator, 52
- /= (forward slash equal) assignment operator, 54
- > (greater than) comparison operator, 57
- >= (greater than or equal to) comparison operator, 57
- < (less than) comparison operator, 57
- <= (less than or equal to) comparison operator, 57
- = (minus equal) assignment operator, 54
- (minus sign)
 - decrement operator (--), 55-56
 - negative numbers, 48
 - subtraction operator, 52
- != (not equal) comparison operator, 57, 88
- () (parentheses)
 - arguments, 68
 - grouping expressions, 59-60
- % (percent sign) modulus operator, 52
- . (period)
 - accessing methods and variables, 59
 - dot notation, 73
- | (pipe character) OR operators, 58
- += (plus equal) assignment operator, 54
- + (plus sign)
 - addition operator, 52
 - concatenation operator, 82

- increment operator (++), 55-56
 - string concatenation, 60-61
- ? (question mark) in SQL statements, 514
- "" (quotation marks) in arguments, 137
- ; (semicolon) statement termination character, 38
- / (slash character), XML tags, 401
- [] (square brackets), arrays, 59, 96
- _ (underscore) in large number literals, 48
- 2D graphics. *See* Java2D

A

- absolute component placement, 334
- abstract classes, 169
- abstract methods, 169, 187
- abstract modifier, 158, 169
- Abstract Windowing Toolkit (AWT). *See* java.awt package
- accept() method, 479
- access control, 159
 - accessor methods, 164
 - comparison of types, 163
 - default access, 159, 175
 - inheritance, 163
 - interfaces, 179
 - packages, 175
 - private access, 159-161
 - protected access, 162
 - public access, 161, 175
- Access databases, Java DB versus, 523
- accessing
 - array elements, 98-99, 233
 - class methods, 165
 - class variables, 75, 165
 - databases, 505
 - instance variables, 72-73
- accessor methods, 164, 187
- action events, 340, 345-346
- ActionListener interface, 330, 340, 345, 460
- actionPerformed() method, 330, 342, 345
- activities (Android apps), 585
- acyclic gradients, 378
- adapter classes, 357-359, 456-457
- addActionListener() method, 330, 341, 345
- addAttribute() method, 533
- addFocusListener() method, 341
- addHandler() method, 560
- adding
 - Apache XML-RPC to NetBeans, 555
 - child nodes to parent nodes, 533
 - classes to packages, 175
 - components
 - to containers, 256, 262-263
 - to panels, 325
 - to toolbars, 298
 - JavaDB library to projects, 512
 - separators to menus, 304
 - stack elements, 239
 - white space to XML documents, 540-542
 - XOM to NetBeans, 532
- addItemListener() method, 341
- addItem() method, 275
- addition operator, 52
- add() method
 - array lists, 233-234
 - border layouts, 323
 - card layouts, 326
 - check boxes/ radio buttons, 273
 - containers, 262
 - menus, 304
- addMouseListener() method, 341
- addMouseMotionListener() method, 341
- addSeparator() method, 304
- addTab() method, 307
- addTextListener() method, 341
- addWindowListener() method, 341
- adjustment events, 340
- AdjustmentListener event listener, 340
- Advogato, 554
- afterLast() method, 521
- aligning
 - components
 - border layouts, 322-324
 - box layouts, 317-319
 - card layouts, 325-333
 - flow layouts, 315-317
 - grid layouts, 320-321
 - panels, 325
 - labels, 267
- AllCapsDemo.java, 442
- allocate() method, 485
- allocating memory, 71
- all-permissions tag, 406
- Alphabet.java, 316
- ampersand (&), AND operators, 57
- AND operators, 57
- Android, 569
 - apps, 569
 - closing projects in Android Studio, 579
 - configuring manifest files in Android Studio, 581
 - creating projects in Android Studio, 572-574, 579
 - designing GUI in Android Studio, 581-584
 - installing and configuring Android Studio, 571-572
 - organizing projects in Android Studio, 574-575
 - preparing resources in Android Studio, 579-580

- running, 577-578, 589-591
 - strings in, 575-577
 - troubleshooting, 578, 610-614
 - writing in Android Studio, 575-577
 - writing Java code in Android Studio, 584-591
- history of, 570-571
- versions of, 592
- android.content package, 587
- Android Developer site, 570
- AndroidManifest.xml, 579-581
- Android SDK Manager, 592
 - installing HAXM, 611-612
- Android Software Development Kit (SDK), 570
- Android Studio, 13, 570
 - closing projects, 579
 - configuring manifest files, 581
 - creating projects, 572-574, 579
 - designing GUI, 581-584
 - installing and configuring, 571-572
 - organizing projects, 574-575
 - preparing resources, 579-580
 - running apps in, 577-578, 589-591
 - troubleshooting, 578, 610
 - checking BIOS settings, 614
 - installing HAXM, 611-613
 - writing apps in, 575-577
 - writing Java code in, 584-591
- android.support.v7.app package, 586
- angle brackets (<>), diamond operator, 247
- anonymous inner classes, 454-459, 466
- antialiasing, 372
- Apache Derby, 503
- Apache Project class libraries, 279
- Apache XML-RPC
 - clients, 556-559
 - data types supported, 565
 - installing, 554-556
 - servers, 559-564
- AppCompatActivity class, 586
- appendChild() method, 533
- append() method, 269
- AppInfo.html, 645
- AppInfo.java, 644
- AppInfo2.java, 647
- applet-desc tag, 413
- applets, 391
 - converting to applications, 413
 - debugging, 655
 - Java Web Start applications versus, 393
 - linking, URL objects, 471
- appletviewer browser, 642-646
- application-desc tag, 403
- applications (Java), 136
 - arguments
 - handling, 138-139
 - passing to, 137
 - Buttons.java, 263
 - converting applets to, 413
 - creating, 135-136
 - debugging, 653-655
 - deployment
 - configuring web servers for Java Web Start, 405
 - creating JNLP files, 396-404
 - description tag, 406
 - icon tag, 406-407
 - Java Web Start, 392-395
 - JNLP security, 405-406
 - digital signatures, 404
 - multitasking. *See* threads
- performance
 - improvements, 407-412
- running, 602-603, 639
- server applications
 - designing, 480-482
 - testing, 482-483
- splash screens, 407
- Storefront, 181-187
- Swing
 - creating interface, 257-259
 - developing framework, 260-261
- threaded
 - example, 213-217
 - writing, 211-213
- apps (Android), 569
 - closing projects in Android Studio, 579
 - configuring manifest files in Android Studio, 581
 - creating projects in Android Studio, 572-574, 579
 - designing GUI in Android Studio, 581-584
 - installing and configuring Android Studio, 571-572
 - organizing projects in Android Studio, 574-575
 - preparing resources in Android Studio, 579-580
 - running, 577-578, 589-591
 - strings in, 575-577
 - troubleshooting, 578, 610
 - checking BIOS settings, 614
 - installing HAXM, 611-613
 - writing in Android Studio, 575-577
 - writing Java code in Android Studio, 584-591
- Arc2D.Float class, 382-383
- archiving files, 650-652
- arcs, drawing with Arc2D.Float class, 382-383

arguments

- command-line
 - arguments, 638
 - setting, 109
 - storing, 111
 - troubleshooting, 139
- creating objects, 68
- grouping, 137
- handling in applications, 138-139
- passing
 - to applications, 137
 - to methods, 132-134
- quotation marks in, 137
- running programs, 623
- in XML-RPC, 553

argument tag, 403

arithmetic, string, 60-61

arithmetic operators, 52-54

ArrayCopier.java, 117

array data type (XML-RPC), 550

ArrayIndexOutOfBoundsException exception, 194

ArrayList class, 227, 233-235

ArrayList() constructor, 556

ArrayList object, 556

array lists

- accessing elements, 233
- creating, 233
- looping through, 235-238

arrays, 96, 226. *See also* loops

- Boolean arrays, data structures versus, 251
- boundaries, 99
- of bytes, casting objects to, 565
- of command-line arguments, 111
- compilation errors, 99
- elements
 - accessing, 98-99
 - changing, 99-102
 - data types, 98
 - in grids, 102
- limitations, 226
- multidimensional, 102

- objects, creating, 97-98
- references, 99
- subscripts, 98-99
- troubleshooting, 99
- variables, declaring, 96-97

arrow operator (->), 460

ASCII character set, 437, 487

assigning values to variables, 40, 43, 62

assignment operators, 54-55

- equal sign (=), 40, 43

associating

- components with event listeners, 341-342
- filters, 421
- .java files with text editor, 630
- MIME types, 405

asterisk (*) multiplication operator, 52

asterisk equal (*=) assignment operator, 54

Atom, 528, 546

attributes, 17-18

- in class hierarchies, 29
- creating, 533
- defining, 17-18
- XML tags, 401, 527

Authenticator.java, 269

Authenticator2.java, 272

author contact information, 608

@author tag (javadoc), 647

autoboxing, 87, 247

AUTOEXEC.BAT, 627-628

Averager.java, 138

AWT (Abstract Windowing Toolkit). *See* java.awt package

B

- background color, setting, 377
- base-2 numbering system, 48
- base-8 numbering system, 48
- base-16 numbering system, 49

- base64 data type (XML-RPC), 550
- BasicStroke class, 380-381
- beforeFirst() method, 521
- behavior, 18-19
 - shared, 31-32
- binary numbers, 48
- BIOS settings, checking, 614
- bits, 227-232
- BitSet class, 227-232
- bitwise operators, 59
- block statements, 38, 103
 - scope, 121
 - try and catch, 196-199
 - finally clause, 199-202
- book website, 607
- Boolean arrays, data structures versus, 251
- boolean data type, 42
 - casting, 83
 - XML-RPC, 550
- Boolean literals, 49
- Border.java, 323
- BorderLayout class, 298, 322
- BorderLayout() constructor, 322
- border layout manager, 322-324
- boundaries, arrays, 99
- Box.java, 141
- Box2.java, 146
- BoxLayout class, 317
- box layout manager, 317-319
- braces. *See* curly braces ({})
- brackets. *See* square brackets ([])
- breaking loops, 119
- break keyword, 107, 119-120
- breakpoints, 652
 - deleting, 654
 - setting, 653-655
- browser (appletviewer), 642-646
- BufferConverter.java, 490
- BufferDemo.java, 429
- buffered character streams
 - reading, 438
 - writing, 440

BufferedInputStream
 class, 427
 BufferedInputStream()
 constructor, 428
 BufferedOutputStream
 class, 427
 BufferedOutputStream()
 constructor, 428
 BufferedReader class, 438
 BufferedReader()
 constructor, 438
 buffered streams, 427-431
 BufferedWriter class, 440
 BufferedWriter()
 constructor, 440
 buffers, 427, 484-486
 byte buffers, 486
 channels, 488-491
 character sets, 487-488
 nonblocking I/O network
 connections, 492-499
 Builder class, 536
 Builder() constructor, 536
 build() method, 536
 built-in fonts, 371
 Bunch.java, 320
 ButtonFrame.java, 262
 ButtonGroup object, 273
 buttons, 255, 261
 differentiating in mouse
 events, 362
 event handling
 action events, 345-346
 item events, 349-351
 fonts, changing, 281
 ImageButton widgets
 (Android), 582-584
 Buttons.java, 263
 ByteBuffer object, 489
 byte buffers, 486-488
 bytecode, 11, 639
 byte data type, 42, 83
 byte filters, 427
 ByteReader.java, 424
 bytes
 arrays of, casting objects
 to, 565

 multiple, writing, 425
 unsigned, 434
 byte streams, 419-422
 file input streams, 422-425
 file output streams,
 425-427
 ByteWriter.java, 426

C

C programs, reading, 444
 C++, Java versus, 11
 Calculator.java, 347
 calling
 constructors, 144, 151
 from another
 constructor, 145-146
 methods, 18, 75-77
 class methods, 80
 nesting calls, 78-79
 in superclasses, 150
 CardLayout class, 326
 card layout manager, 325-333
 cards, 326
 caret (^), XOR operator, 58
 case keyword, 107
 case-sensitivity of Java, 41
 casting. *See also* converting
 definition of, 83
 destinations, 83
 explicit casts, 84
 objects, 82-85, 565
 primitive types, 82-84
 sources, 83
 catching exceptions, 194-196
 try and catch blocks,
 196-199
 finally clause, 199-202
 CD command (MS-DOS),
 621-622
 certificate authorities, 404
 chaining methods, 655
 changing. *See also* modifying
 array elements, 99-102
 button fonts, 281
 channel() method, 495
 channels, 488-499
 character buffers, converting
 to byte buffers, 487-488
 character encodings, 541
 character literals, 49-50
 character sets
 buffers, 487-488
 Unicode, 40
 escape codes, 49-50
 character streams,
 419-420, 437
 reading text files, 437-440
 writing text files, 440-441
 charAt() method, 77, 91
 char data type, 42
 casting, 83
 int data type versus, 445
 Charset class, 487
 CharsetDecoder class, 487
 CharsetEncoder class, 487
 charWidth() method, 373
 check boxes, 255, 272-274
 event handling
 action events, 345-346
 item events, 349-351
 nonexclusive, 273
 checked exceptions, 204, 195
 child nodes, adding to parent
 nodes, 533
 .class extensions, 639
 classes, 11, 14-16. *See also*
 packages
 abstract classes, 169
 adapter classes, 357-359,
 456-457
 adding to packages, 175
 AppCompatActivity, 586
 Arc2D.Float, 382-383
 ArrayList, 227, 233-235
 attributes, 17-18
 BasicStroke, 380-381
 behavior, 18-19
 BitSet, 227-232
 BorderLayout, 298, 322
 BoxLayout, 317
 BufferedInputStream, 427

- BufferedOutputStream, 427
- BufferedReader, 438
- BufferedWriter, 440
- Builder, 536
- CardLayout, 326
- Charset, 487
- CharsetDecoder, 487
- CharsetEncoder, 487
- Color, 32, 375
- ColorSpace, 375
- compiling, 21, 601
- Component, 335
- ConfirmDialog, 286-288
- constants, 43-44
- Container, 256
- creating, 19-22, 600-602
- Cursor, 461
- DataInputStream, 422
- DataOutputStream, 422
- defining, 126
- Dictionary, 227
- Dimension, 258
- DriverManager, 509
- Ellipse2D.Float, 382
- Error, 194
- Exception, 194-195, 198
- exception classes, 208
- File, 441
- FileInputStream, 422
- FileOutputStream, 422
- FileReader, 437
- Files, 442
- FileSystems, 441
- FileWriter, 440, 445
- FilterInputStream, 427
- FilterOutputStream, 427
- final classes, 167-169
- FlowLayout, 314-315
- FocusAdapter, 358, 457
- FontMetrics, 373
- Graphics, 368
- Graphics2D, 368
 - coordinate system, 369-370
 - creating drawing surface, 368-369
- GridLayout, 320
- grouping, 32
- HashMap, 227, 241-246
- helper classes, 136, 450
- hierarchies, 26-27
 - creating, 27-29
 - methods in, 30
- URLConnection, 474
- identifying, 170
- importing, 171-173
- InetAddress, 493
- InetSocketAddress, 493
- inheritance, 176
- inner classes, 359-362, 388, 450-453
 - anonymous inner classes, 454-459, 466
 - creating, 450
 - scope, 450
- InputDialog, 286-289
- InputStream, 422
- InputStreamReader, 437
- Insets, 334
- instances of, creating, 16
- Integer, 86
- Intent, 587
- interfaces versus, 176. *See also* interfaces
- IOException, 195
- Java Class Library, 16, 159, 225, 278-281
- JButton, 16, 261
- JCheckBox, 272
- JComboBox, 274-275
- JComponent, 256, 264
- JFrame, 257
- JLabel, 267
- JList, 276
- JMenu, 303
- JMenuBar, 303
- JMenuItem, 303
- JOptionPane, 286
- JPanel, 262, 325, 368
- JPasswordField, 268
- JProgressBar, 300
- JRadioButton, 272
- JScrollBar, 297
- JScrollPane, 271, 296
- JSlider, 294
- JTabbedPane, 307
- JTextComponent, 268
- JTextField, 268
- JToggleButton, 272
- JToolBar, 297
- Key, 388
- KeyAdapter, 358, 457
- libraries. *See* libraries
- Line2D.Float, 381
- listeners. *See* listeners
- loading, 508
- main classes, designating, 136
- Math, 79, 280
- MessageDialog, 286, 289
- methods, 18
- MouseAdapter, 358, 457
- MouseMotionAdapter, 358
- name conflicts, 172-173
- NamedPoint, 151
- Node, 533
- Object, 26
- objects, casting, 84-85
- object wrappers, 86
- of objects, determining, 89-90
- OptionDialog, 286, 290-291
- organizing, 25, 158, 170
 - creating hierarchies, 27-29
 - inheritance, 25-31
 - interfaces, 31-32
 - packages, 32, 45
- OutputStream, 422
- OutputStreamWriter, 440
- PreparedStatement, 514
- PrintStream, 79
- programs versus, 125
- PropertyHandlerMapping, 560
- protecting, 170
- Reader, 437
- Rectangle2D.Float, 382
- RenderingHint.Key, 388

- R.java, 586
- RuntimeException, 195
- SelectionKey, 494
- Serializer, 540
- ServerSocket, 479
- Socket, 475
- SocketChannel, 493
- SocketImpl, 480
- Stack, 227, 238-239
- String, 79, 201
- StringTokenizer, 69
- subclasses, 25-26
- superclasses, 25
 - indicating, 126
 - modifying, 27
- SwingWorker, 407-413
- System, 79, 657
 - class methods, 134
 - in variable (input stream), 431
- Text, 533
- Thread, 191, 211
- Throwable, 194-195
- TimeZone, 657
- UIManager, 259
- URL, 471
- variables, 126-127
- Vector, 235
- WebServer, 559
- WindowAdapter, 358, 456
- wrapper classes, 134
- Writer, 437
- XmlRpcClient, 556
- XmlRpcServer, 559
- classes command (jdb), 656
- class files, specifying, 640
- class keyword, 126, 450
- class methods, 19, 79-80, 134-135
 - accessing, 165
 - calling, 80
 - defining, 134
- Class not found error, 633-635
- CLASSPATH variable (MS-DOS)
 - Windows 7-10, 633-635
 - Windows 98/Me, 635-636
- class types, 43
- class variables, 18, 39, 72, 127
 - accessing, 165
 - defining, 74
 - initial values, 40
 - instance variables versus, 33, 74
 - troubleshooting, 75
 - values, accessing/modifying, 75
- clear command (jdb), 654
- clear() method
 - array lists, 235
 - hash maps, 242
- clients (XML-RPC), 556-559
- client-side sockets
 - closing, 476
 - nonblocking clients, 493-499
 - opening, 475
- Clone command (appletviewer), 644
- close() method
 - buffered character streams, 441
 - character streams, 440
 - client-side sockets, 476
 - data source connections, 512
 - data streams, 434
 - file output streams, 425
 - streams, 420-421
- closePath() method, 384
- closing
 - data source connections, 512
 - frames, 259-260
 - projects in Android Studio, 579
 - socket connections, 476
- closing tags (XML), 401, 527
- ClosureMayhem.java, 464
- closures, 449, 460-466
- codebase attribute, 402
- CodeKeeper.java, 236
- CodeKeeper2.java, 248
- code listings. *See* listings
- code signing, 658-659
- color, 375
 - background colors, 377
 - Color objects, creating, 376
 - dithering, 375
 - drawing colors, setting, 376-377
 - finding current color, 377
 - sRGB color system, 375
 - XYZ color system, 375
- Color class, 32, 375
- Color objects, creating, 376
- ColorSpace class, 375
- color spaces, 375
- combining layout managers, 324-325
- combo boxes, 274-276
 - action events, 345-346
 - item events, 349-351
- ComicBooks.java, 243
- ComicBox.java, 452
- command line, 12, 638-639
- command-line arguments
 - setting, 109
 - storing, 111
 - troubleshooting, 139
- command-line interfaces, 619-621
- command-line tools, javac, 631
- commands. *See also* keywords; statements
 - import, 32-34
 - java -version, 624
 - jdb (debugger)
 - !!, 654
 - classes, 656
 - clear, 654
 - cont, 654
 - down, 655
 - exit, 655
 - ignore, 656
 - list, 654
 - locals, 654
 - memory, 656
 - methods, 656

- print, 654
- run, 654
- step, 654
- stop at, 653
- stop in, 653
- suspend, 656
- threads, 656
- up, 655
- menu commands,
 - appletviewer browser, 643-644
- MS-DOS
 - CD, 621-622
 - CLASSPATH variable, 633-636
 - MD, 622
 - PATH variable, 625-628
- comments, 46
 - notation, 46-47
 - in source code, 646
- Comparable interface, 32
- comparing
 - objects, 87-89
 - strings, 88-89
- comparison operators, 56-57, 88
- compiler errors, 210
 - about generics, 251
 - for arrays, 99
 - runtime errors versus, 247
- compilers, 21, 641-642
- compiling
 - classes, 21, 601
 - files, 641-642
 - Java programs in Windows, 631-632
 - troubleshooting, 601, 633
- Component class, 335
- components. *See also names of specific components*
 - associating with event listeners, 341-342
- Swing, 256, 264
 - absolute placement, 334
 - adding to containers, 256, 262-263
 - adding to panels, 325
- AWT components
 - versus, 256
- check boxes, 272-274
- combo boxes, 274-276
- creating, 256, 261-262
- dialog boxes, 286-293
- disabled, 264
- drop-down lists, 274-276
- hiding, 264
- image icons, 265-267
- labels, 267
- layout managers. *See* layout managers
- lists, 276-278
- menus, 303-307
- progress bars, 300-303
- radio buttons, 272-274
- resizing, 264
- scrolling panes, 271-272, 296-297
- sliders, 294-296
- tabbed panes, 307-310
- text areas, 269-271
- text fields, 268
- toolbars, 297-300
- windows, frames, 257
- concatenating strings, 60-61
- concatenation operator (+), 82
- conditional operator. *See* ternary operator
- conditionals
 - if, 104-106
 - switch, 105-111, 121
 - ternary operator, 112
- configureBlocking()
 - method, 493
- configuring
 - Android Studio, 571-572
 - Java Development Kit (JDK), 619
 - command-line interface, 619-621
 - creating folders, 622-623
 - opening folders, 621-622
 - running programs, 623
 - setting CLASSPATH variable, 633-636
 - setting PATH variable, 624-628
 - manifest files in Android Studio, 581
 - scrollbars, 271
 - web servers for Java Web Start applications, 405
- confirm dialog boxes, 287-288
- ConfirmDialog class, 286-288
- conflicts, name
 - classes, 172-173
 - reducing, 170
- connecting
 - to databases, 505-510
 - troubleshooting, 514
 - viewing connection information, 510
 - to Internet. *See* networking
- connect() method, 493
- consistency checking (exceptions), 195-196
- ConsoleInput.java, 432
- console. *See* command line
- console input streams, 431-432
- constant variables. *See* final variables
- constants, 43
 - declaring, 44
 - enumerations, 249-250
 - naming, 44
- constructors, 69, 144-145
 - ArrayList(), 556
 - BorderLayout(), 322
 - BufferedInputStream(), 428
 - BufferedOutputStream(), 428
 - BufferedReader(), 438
 - BufferedWriter(), 440
 - Builder(), 536
 - calling, 144-146, 151
 - DataInputStream(), 433

- DataOutputStream(), 433
- definition of, 71
- Dimension(), 258
- exception classes, 208
- FileInputStream(), 422
- FileOutputStream(), 425
- FileReader(), 437
- FileWriter(), 440
- FlowLayout(), 315
- GridLayout(), 320
- Intent(), 587
- JCheckBox(), 272
- JComboBox(), 275
- JFrame(), 257
- JList(), 276
- JMenuBar(), 305
- JMenuItem(), 303
- JProgressBar(), 301
- JScrollPane(), 271, 296
- JSlider(), 294
- JTabbedPane(), 307
- JTextArea(), 269
- JTextField(), 268
- JToolBar(), 298
- naming, 144
- overloading, 146-147
- overriding, 150-152
- Serializer(), 541
- URL(), 471
- WebServer(), 559
- Container class, 256
- containers, 255
 - absolute component placement, 334
 - adding components to, 256, 262-263
 - cards, 326
 - layout managers. *See* layout managers
 - menus, creating, 304
 - panels, 262, 325-326
- contains() method, 235
- containsKey() method, 242
- containsValue() method, 242
- cont command (jdb), 654
- continue keyword, 119-120
- controlling access. *See* access control
- converting. *See also* casting
 - applets to applications, 413
 - character and byte buffers, 487-488
 - primitive types and objects, 86-87
 - source code, 641
 - strings to numbers, 86
- coordinate systems
 - Java2D, 369-370
 - user versus device coordinate spaces, 378
- Cover Pages, 530
- createFont() method, 371
- createStatement() method, 510, 522
- creating. *See also* constructors
 - array lists, 233
 - array objects, 97-98
 - attributes, 533
 - buffered input streams, 428
 - buffered output streams, 428
 - character sets, 487
 - classes, 19-22, 600-602
 - components, Swing, 256, 261-262
 - confirm dialog boxes, 287-288
 - database tables, 517
 - data input streams, 433
 - data output streams, 433
 - drawing surfaces, 368-369
 - exceptions, 207
 - file input streams, 422
 - File objects, 441
 - file output streams, 425
 - folders in MS-DOS, 622-623
 - frames, 368
 - frameworks (GUI), 260-261
 - hash maps, 241
 - inner classes, 450
 - input dialog boxes, 288-289
 - input streams, 420
 - instances, 16
 - intents, 587
 - interfaces, 178-179
 - interfaces (GUI), 257-259
 - Java applications, 135-136, 629-631
 - JNLP files, 396-404
 - labels, 267
 - layout managers, 314
 - menu containers, 304
 - message dialog boxes, 289
 - methods, overloaded, 140-143
 - objects, 68
 - arguments, 68
 - with closures, 460-465
 - Color, 376
 - with constructors, 71
 - Document, 533
 - Element, 533
 - Font, 370-372
 - GeneralPath, 384
 - ImageIcon, 265
 - with new operator, 68-70
 - Serializer, 540
 - String, 21
 - StringTokenizer objects, 69-70
 - URL, 471
 - online storefronts, 181-187
 - option dialog boxes, 290-291
 - output streams, 420
 - overridden methods, 148-149
 - packages, 640
 - panels, 325, 368
 - Path objects, 442
 - projects, 19
 - Android Studio, 572-574, 579
 - NetBeans, 598-600
 - scrolling panes, 296

- Selector objects, 493
- server sockets, 479
- source files, 629
- stacks, 238
- system properties, 657
- threads, 212
- variables, 39-40, 44-45
- XML documents, 532-535
- XML-RPC servers, 559-564
- curly braces (`{}`), 38
 - block statements, 103
- current objects, referring to, 130
- Cursor class, 461
- CursorMayhem.java, 462
- cursors, 461
- CustomerReporter.java, 512
- custom packages
 - access control, 175
 - classes, adding, 175
 - folder structure, 174
 - naming, 173-174
- cyclic gradients, 378

D

- data, storing, 427
- databases
 - accessing, 505
 - Apache Derby, 503
 - connecting to, 505-510
 - troubleshooting, 514
 - viewing connection information, 510
- data source connections
 - closing, 512
 - opening, 508-510
- drivers, 504-505, 508
- Java DB, 503, 523
- JDBC. *See* JDBC
- queries, 504, 508-511
- records
 - navigating, 511, 521-522
 - reading, 508-513
 - writing, 514-521

- tables
 - creating, 517
 - viewing, 507-508
 - for XML-RPC servers, 564
- DataInputStream class, 422
- DataInputStream()
 - constructor, 433
- data input streams, 433
- DataOutputStream class, 422
- DataOutputStream()
 - constructors, 433
- data output streams, 433
- data source connections
 - closing, 512
 - opening, 508-510
- data streams, 433-436
- data structures, 226
 - ArrayList class, 227, 233-235
 - arrays. *See* arrays
 - BitSet class, 227-232
 - Boolean arrays versus, 251
 - Enumeration interface, 227
 - generics, 246-250
 - HashMap class, 227, 241-246
 - Iterator interface, 227-229, 235-238
 - looping through, 235-238
 - Map interface, 240-241
 - Stack class, 227, 238-239
 - Vector class, 235
- data types, 42-43
 - Apache XML-RPC support, 565
 - array elements, 98
 - boolean values, 42
 - casting, 83-84
 - characters, 42
 - char versus int data types, 445
 - enumerations, 249-250
 - floating-point numbers, 42
 - integers, 42
 - objects versus, 91
 - primitive, 42-43

- to remote methods, 557
- void, 43
- XML-RPC support, 550
- dateTime.iso8601 data type (XML-RPC), 550
- DayCounter.java, 108
- deallocating memory, 71
- debugger (jdb), 642, 652-653
 - advanced commands, 655-656
 - applet debugging, 655
 - application debugging, 653-655
- debuggers (XML-RPC), 554
- debugging, 652. *See also* troubleshooting
 - advanced commands, 655-656
 - applets, 655
 - applications, 653-655
 - breakpoints, 652
 - deleting, 654
 - setting, 653-655
 - single-step execution, 652
- declarations
 - import, 171-172
 - package, 175
- declaring
 - array variables, 96-97
 - arrays of arrays, 102
 - constants, 44
 - interfaces, 176-179
 - variables, 39-40
- decode() method, 488
- decrementing variables, 55-56
- decrement operator (`-`), 55-56
- default access, 159
 - packages, 175
 - protected access versus, 162
- default package, 63
- defining
 - attributes, 17-18
 - classes, 126
 - hierarchies, 169
 - instance variables, 21

- methods, 21, 128-130
 - class methods, 134
 - this keyword, 130-131
 - variable scope, 131-132
- subclasses, 26
- values, shared, 43
- variables
 - class variables, 74
 - instance variables, 126-127
- delete() method, 442
- deleting
 - breakpoints, 654
 - files, 442
- deploying applications (Java Web Start), 392-395
 - configuring web servers for, 405
 - creating JNLP files, 396-404
 - description tag, 406
 - icon tag, 406-407
 - security, 405-406
- @deprecated tag (javadoc), 650
- deprecated methods, 642
- description tag, 406
- designing. *See also* creating
 - GUI in Android Studio, 581-584
 - server applications, 480-482
 - XML dialects, 528-529
- destinations (casting), 83
- development tools, selecting, 12-13, 616
- device coordinate space, 378
- dialects (XML), designing, 528-529
- dialog boxes, 286
 - confirm dialog boxes, 287-288
 - example, 291-293
 - input dialog boxes, 288-289
 - message dialog boxes, 289
 - option dialog boxes, 290-291
- diamond operator, 247
- DiceRoller.java, 410
- DiceWorker.java, 408
- Dictionary class, 227
- differentiating buttons in mouse events, 362
- digital signatures, 404
- Dimension class, 258
- Dimension() constructor, 258
- Dimension object, 264, 296
- dimens.xml, 579
- disabled components, 264
- displaying frames, 258
- dithering, 375
- division operator, 52
- DmozHandlerImpl.java, 563
- DmozHandler.java, 562
- DmozServer.java, 561
- dockable toolbars, 298
- !DOCTYPE declaration, 529
- documentation
 - Java Class Library, 279
 - viewing, 47
- documentation tool (javadoc), 646-650
- Document object, creating, 533
- Document Object Model (DOM), 530
- documents
 - HTML, viewing, 643
 - XML
 - creating, 532-535
 - formatting, 540-542
 - modifying, 536-540
- Document Type Definition (DTD), 529
- doInBackground() method, 408
- dollar sign (\$) in variable names, 41
- do loops, 118-119
- DomainEditor.java, 538
- DomainWriter.java, 541
- DOM (Document Object Model), 530
- dot notation, 72-73
 - calling class methods, 80
 - calling methods, 75
 - evaluating, 73
- double ampersand (&&), AND operators, 57
- double data type, 42
 - casting, 83
 - XML-RPC, 550
- double pipe character (| |), OR operators, 58
- down command (jdb), 655
- downloading
 - Apache XML-RPC, 554
 - JDK, 638
- drawing
 - lines, 377
 - Line2D.Float class, 381
 - rendering attributes, 378-381
 - maps, 385-387
 - objects, 384
 - polygons, 377, 383-384
 - arcs, 382-383
 - ellipses, 382
 - rectangles, 381
 - rendering attributes, 378-381
 - strokes, 380-381
 - text, 370-372
 - antialiasing, 372
 - finding font information, 372-375
- drawing colors, setting, 376-377
- drawing surfaces, creating, 368-369
- draw() method, 384
- drawString() method, 370
- DriverManager class, 509
- drivers
 - for databases, 504-505, 508
 - USB, installing, 591

drop-down lists, 255, 274-276
 DTD (Document Type Definition), 529
 dynamic garbage collection, 72

E

Eclipse website, 13
 editing. *See also* changing; modifying
 system properties, 656-658
 XML files, 576-577
 editors (text). *See* text editors
 Element object, creating, 533
 elements (arrays)
 accessing, 98-99, 233
 changing, 99-102
 data types, 98
 grids, 102
 elements (stack)
 adding, 239
 popping off, 239
 searching, 239
 elements (XML). *See* tags (XML)
 Ellipse2D.Float class, 382
 ellipses, drawing, 382
 else keyword, 104
 email address of author, 608
 EML (Extended Machine Language), 546
 empty() method, 239
 empty statements in loops, 114
 emulators. *See* Android Studio
 enabling Intel Virtualization Technology in BIOS settings, 614
 encapsulation, 159-161
 enclosure tag, 528
 encode() method, 488
 endcap styles (drawing strokes), 380
 ending. *See* stopping
 end-of-line characters, 438, 441
 Enumeration interface, 227
 enumerations, 249-250
 enum keyword, 249
 environment variables, 656
 EOFException (end-of-file exception), 195
 data streams, 434
 I/O streams, 422
 equal sign (=) assignment operator, 40, 43, 54-55
 equals() method, 88, 242
 EqualsTester.java, 88
 equal symbol (==) comparison operator, 57, 88
 Error class, 194
 error-handling. *See also* errors
 catching exceptions, 196
 finally clause, 199-202
 try and catch blocks, 196-199
 consistency checking, 195-196
 passing exceptions, 204-205
 throwing exceptions, 202, 207
 checked, 204
 inheritance, 206
 nested handlers, 208-209
 throws clause, 203
 unchecked, 204
 traditional method, 192-193
 errors, 218. *See also* debugging; error-handling; exceptions; troubleshooting
 Class not found, 633-635
 compiler errors, 210
 about generics, 251
 for arrays, 99
 runtime errors versus, 247
 Error class, 194
 Exception class, 194
 fatal, troubleshooting, 196
 NoClassDef, 633-635
 runtime errors, compiler errors versus, 247
 escape codes (Unicode character set), 49-50
 evaluating
 dot notation, 73
 XOM, 542-545
 event-handling, 339. *See also* event listeners
 action events, 345-346
 components, associating with event listeners, 341-342
 focus events, 346-349
 item events, 349-351
 keyboard events, 351-352
 methods, 342-345
 mouse events, 352
 mouse movement events, 352-357
 window events, 357
 event listeners, 330, 340
 ActionListener, 340, 345
 adapter classes and, 357-359
 AdjustmentListener, 340
 associating components with, 341-342
 FocusListener, 340, 346
 importing, 341
 inner classes and, 359-362
 ItemListener, 340
 KeyListener, 340, 351
 MouseListener, 340, 352
 MouseMotionListener, 340, 352
 property change listeners, 409
 WindowListener, 340, 357, 455
 events
 action events, 340, 345-346
 adjustment events, 340
 focus events, 346-349
 item events, 340, 349-351

- keyboard events, 340, 351-352
- keyboard focus events, 340
- mouse events, 340, 352, 362
- mouse movement events, 340, 352-357
- window events, 340, 357
- example code. *See* listings
- Exception class, 194-195, 198
- exception classes, constructors, 208
- @exception tag (javadoc), 650
- exceptions, 191-195. *See also* debugging; error-handling; errors; troubleshooting
 - ArrayIndexOutOfBoundsException, 194
 - catching, 194-196
 - finally clause, 199-202
 - try and catch blocks, 196-199
 - checked, 195
 - compiler errors, 210
 - consistency checking, 195-196
 - creating, 207
 - EOFException, 195, 422, 434
 - Error class, 194
 - Exception class, 194-195
 - file operations, 442
 - inheritance, 207
 - InterruptedException, 475
 - IOException, 195, 489, 536
 - I/O streams, 421-422
 - limitations, 210
 - MalformedURLException, 195, 471
 - non-runtime, 219
 - NullPointerException, 194
 - ParseException, 536
 - passing, 204-205
 - runtime, 194, 219
 - RuntimeException, 195

- SQLException, 509-511
- Throwable class, 194-195
- throwing, 194, 202, 207
 - checked exceptions, 204
 - inheritance issues, 206
 - nested handlers, 208-209
 - throws clause, 203
 - unchecked exceptions, 204
- exclamation point (!), NOT operator, 58
- exclusive radio buttons, 273
- execute() method, 557
- executeQuery() method, 510-511
- exit command (jdb), 655
- exiting loops, 119
- expanding NetBeans panes, 604
- explicit casts, 84
- exponents in floating-point literals, 48
- expressions, 51-52
 - definition of, 38
 - dot notation, 73
 - grouping, 59
 - operators. *See* operators
 - readability, improving, 60
 - return values, 38, 52
- extending interfaces, 180-181
- extends keyword, 126, 180
- Extensible Markup Language. *See* XML
- extensions
 - .class, 639
 - .java, 641

F

- false value (Boolean), 49
- fatal errors, troubleshooting, 196
- feed2.rss, 540
- FeedBar.java, 299
- FeedBar2.java, 305

- FeedInfo.java, 291
- FileChannel objects, 488
- File class, 441
- file extensions. *See* extensions
- FileInputStream class, 422
- FileInputStream() constructor, 422
- file input streams, 422-425
- FileNotFoundException, 421
- File objects, creating, 441
- FileOutputStream class, 422
- FileOutputStream() constructor, 425
- file output streams, 425-427
- FileReader class, 437
- FileReader() constructor, 437
- files
 - archiving, 650-652
 - compiling, 641-642
 - deleting, 442
 - JAR files, signing, 658-659
 - JNLP files
 - associating MIME types, 405
 - creating, 396-404
 - description tag, 406
 - icon tag, 406-407
 - security, 405-406
 - multiple, compiling, 641
 - Path object, 441-444
 - relative paths, 444
 - renaming, 442
 - text files
 - reading, 437-440
 - writing, 440-441
 - XML files, editing, 576-577
- Files class, 442
- FileSystems class, 441
- FileWriter class, 440, 445
- FileWriter() constructor, 440
- fill() method, 384
- fill patterns (Java2D), 378-380
- filtering streams, 421, 427
- FilterInputStream class, 427
- FilterOutputStream class, 427
- filters (streams), 421

- final abstract methods, 187
- final classes, 167-169
- final keyword, 44
- final methods, 167-168
- final modifier, 158, 167
- final variable, 167
- finally statement, 199-202
- finding font information, 372-375
- Finger.java, 477
- Finger protocol, 476
- FingerServer.java, 495
- finishConnect() method, 495
- first() method, 521
- flags, 227-232
- flip() method, 485
- float data type, 42, 83
- floating-point numbers, 42
 - exponents, 48
 - hexadecimal numbers
 - versus, 376
 - Java2D, 388
 - as literals, 48
- floor() method, 280
- FlowLayout class, 314-315
- FlowLayout() constructor, 315
- flow layout manager, 315-317
- flush() method, 428
- FocusAdapter class, 358, 457
- focus events, 346-349
- focusGained() method, 346
- FocusListener event listener, 340, 346
- focusLost() method, 346
- folders
 - MS-DOS
 - creating, 622-623
 - opening, 621-622
 - structure (packages), 174
- FontMetrics class, 373
- Font objects, creating, 370-372
- fonts
 - antialiasing, 372
 - built-in, 371
 - on buttons, changing, 281

- finding information, 372-375
 - Font objects, creating, 370-372
 - styles, selecting, 371
- for loops, 113-116
- format commands (JDK), 638-639
- FormatChooser.java, 349
- FormatFrame.java, 273
- FormatFrame2.java, 275
- formatting
 - strings, 77-78
 - XML documents, 540-542
- forName() method
 - character sets, 487
 - database drivers, 508
- forward slash (/)
 - comment notation, 46
 - division operator, 52
- forward slash equal (/=)
 - assignment operator, 54
- frames, 255-257
 - absolute component
 - placement, 334
 - closing, 259-260
 - creating, 368
 - developing framework, 260-261
 - displaying, 258
 - locations, 258
 - resizing, 310
 - sizing, 257
 - visible, 258
- frameworks (GUI), creating, 260-261
- functional interfaces, 460
- functional programming, 449, 465
- functions of tools, modifying, 638

G

- GeneralPath objects, creating, 384
- generics, 246-251

- getActionCommand() method, 345
- getAppletInfo() method, 644-645, 655
- getChannel() method, 488
- getChar() method, 486
- getChildElements()
 - method, 537
- getChild() method, 537
- getClass() method, 90
- getClickCount() method, 352
- getColor() method, 377
- getConnection() method, 509
- getContentType() method, 474
- getDate() method, 511
- getDefault() method, 441
- getDouble() method
 - byte buffers, 486
 - database records, 511
- getErrorCode() method, 511
- getFirstChildElement()
 - method, 537
- getFloat() method
 - byte buffers, 486
 - database records, 511
- getFontMetrics() method, 373
- getHeaderFieldKey()
 - method, 474
- getHeaderField() method, 474
- getHeight() method, 373
- getIcon() method, 267
- getId() method, 586
- getInsets() method, 334
- getInt() method
 - byte buffers, 486
 - database records, 511
- getItemAt() method, 275
- getItemCount() method, 275
- getItem() method, 349
- getKeyChar() method, 351
- getLong() method
 - byte buffers, 486
 - database records, 511
- getMessage() method, 197
- get() method
 - array lists, 233
 - buffers, 484

- elements, 537
- Map interface, 241
- getParameterInfo() method, 644-645
- getPath() method, 441
- getPoint() method, 352
- getProperties() method, 657
- getProperty() method, 657
- get requests, 551
- getResponseCode() method, 474
- getResponseMessage() method, 474
- getRootElement() method, 536
- getSelectedIndex() method, 275
- getSelectedItem() method, 275
- getSelectedText() method, 268
- getSelectedValuesList() method, 277
- getShort() method, 486
- getSize() method, 264
- getSource() method, 342, 345
- getSQLState() method, 511
- getStateChange() method, 349
- getString() method, 511
- getText() method, 267-268
- getX() method, 352
- getXmlRpcServer() method, 559
- getY() method, 352
- GiftShop.java, 185
- GNU Lesser General Public License (LGPL), 531
- Google, history of Android, 570
- Gosling, James, 9-10, 571, 598
- gradient fills, 378
- graphical user interface. *See* GUI
- graphics. *See also* image icons
 - 2D graphics. *See* Java2D
 - in Android apps, 579
 - Java2D graphics, casting objects, 85

- organizing in NetBeans, 266
- Graphics2D class, 368
 - coordinate system, 369-370
 - creating drawing surface, 368-369
 - drawing objects, 384
- Graphics2D objects, casting, 85
- Graphics class, 368
- Graphics objects, casting, 85
- greater than or equal to symbol ($\geq$) comparison operator, 57
- greater than symbol ($>$) comparison operator, 57
- GridLayout class, 320
- GridLayout() constructor, 320
- grid layout manager, 320-321
- grids, array elements, 102
- grouping
 - arguments, 137
 - classes, 32
 - expressions, 59
 - interfaces, 32
 - methods, 79
 - packages in NetBeans, 183
- guid tag, 527
- GUI (graphical user interface)
 - designing in Android Studio, 581-584
 - Swing. *See* Swing

H

- HalfDollars.java, 100
- HalfLooper.java, 115
- handling
 - arguments in applications, 138-139
 - errors. *See* error-handling
- Hardware Accelerated Execution Manager. *See* HAXM
- hardware requirements for HAXM, 612

- Harold, Elliott Rusty, 531, 545
- hashCode() method, 242
- HashMap class, 227, 241-246
- hash maps, creating, 241
- hasNext() method, 228
- HAXM (Hardware Accelerated Execution Manager), 610
 - checking BIOS settings, 614
 - installing, 611-613
 - requirements, 612
- HelloUser.java, 630
- helper classes, 136, 450
- hexadecimal numbers, 49
 - floating-point literals versus, 376
- HexReader.java, 200
- hiding components, 264
- hierarchies, 26-27
 - creating, 27-29
 - defining, 169
 - interface, 181
 - methods in, 30
- history
 - of Android, 570-571
 - of Java, 10-11
- HolidaySked.java, 231
- homepage tag, 401
- href attribute, 402
- HTML documents, viewing, 643
- HTTP, XML-RPC requests
 - responding to, 553-554
 - sending, 551-552
- URLConnection class, 474
- hyphen (-). *See* minus sign (-)

I

- icon tag, 402, 406-407
- IconFrame.java, 265
- icons
 - for Android apps, 579-581
 - image icons, 255, 265-267

- Java Web Start applications, 399
- IDEA website, 13
- identifying classes, 170
- IDEs (integrated development environments)
 - Android Studio. *See* Android Studio
 - NetBeans. *See* NetBeans
 - selecting, 12-13
- if statements, 104-106
- ignore command (jdb), 656
- ImageButton widgets (Android), 582-584
- image icons, 255, 265-267
- ImageIcon objects, 265
- implementing interfaces, 177-178
- implements keyword, 177, 212
- import declaration, 171-172
- import statement, 32-34, 175, 256
- importing
 - Apache XML-RPC, 555
 - classes, 171-173
 - event listeners, 341
 - packages, 171
- improving
 - application performance, 407-412
 - readability
 - of expressions, 60
 - of programs, 46
- increment operator (++), 55-56
- incrementing variables, 55-56
- increments in loops, 113
- indexOf() method, 77, 235
- index values of loops, 114
- InetAddress class, 493
- InetSocketAddress class, 493
- Info command (appletviewer), 644
- Info.java application, 292
- information tag, 402
- inheritance, 25-31
 - access control, 163
 - class hierarchies, creating, 27-29
 - exceptions, creating, 207
 - multiple, 31, 176
 - single, 31, 176
 - throwing exceptions, 206
- initializing
 - loops, 113
 - objects, 71
- inner classes, 359-362, 388, 450-453
 - anonymous inner classes, 454-459, 466
 - creating, 450
 - scope, 450
- input dialog boxes, 288-289
- InputDialog class, 286-289
- input/output. *See* I/O streams
- InputStream class, 422
- InputStreamReader class, 437
- input streams, 371, 420. *See also* streams
 - buffered input streams, 428
 - console input streams, 431-432
 - creating, 420
 - data input streams, 433
 - file input streams, 422-425
- insertChild() method, 541
- insert() method, 269
- insets, 333-334
- Insets class, 334
- installing
 - Android Studio, 571-572
 - Apache XML-RPC, 554-556
 - HAXM, 611-613
 - JDK (Java Development Kit), 616-619
 - JRE (Java Runtime Environment), 392
 - NetBeans, 598
- InstanceCounter.java, 165
- instance methods, 135. *See also* methods
- instanceof operator, 59, 90, 343
- instances, 15-16. *See also* objects
- instance variables, 17, 39, 72
 - class variables versus, 33, 74
 - defining, 21, 126-127
 - initial values, 40
 - length, 99
 - nesting with method calls, 79
 - values
 - accessing, 72-73
 - modifying, 73-74
- instantiation, 15
- int data type, 42
 - casting, 83
 - char data type versus, 445
 - XML-RPC, 550
- Integer class, 86
- integer literals, 47-49
- integers, data types, 42
- integrated development environments. *See* IDEs
- IntelliJ IDEA, 574
- Intel Virtualization Technology, enabling in BIOS settings, 614
- Intent class, 587
- Intent() constructor, 587
- intents, creating, 587
- interface hierarchy, 181
- interfaces, 31-32, 176
 - access control, 179
 - ActionListener, 330, 460
 - adapter classes, 357-359
 - classes versus, 176
 - command-line, 619-621
 - Comparable, 32
 - creating, 178-179
 - declaring, 176-179
 - Enumeration, 227
 - event listeners. *See* event listeners
 - extending, 180-181
 - functional interfaces, 460

- grouping, 32
- implementing, 177
- Iterator, 227-229, 235-238
- Map, 240-241
- methods, 179-180
- multiple interfaces, implementing, 177-178
- objects, casting, 85
- Paint, 378
- Runnable, 191, 212
- ScrollPaneConstants, 271, 297
- SocketImplFactory, 480
- Statement, 510
- SwingConstants, 267, 294, 309
- as variable type, 178
- variables, 179
- WindowListener, 455
- interfaces (GUI)
 - creating, 257-259
 - event listeners. *See* event listeners
 - interface libraries, 310
 - wizard interface, 327
- Internet Assigned Numbers Authority, 480
- Internet connections. *See* networking
- interpreter (java), 639-641
- InterruptedException
 - errors, 475
- in variable (input stream), 431
- invoking. *See* calling
- I/O (input/output) streams, 419-420
 - buffered, 427-431
 - buffers, 484-486
 - byte streams, 420-422
 - channels, 488-491
 - character sets, 487-488
 - character streams, 420, 437-441
 - console input, 431-432
 - creating, 420
 - data streams, 433-436

- exception handling, 421-422
- file input streams, 422-425
- file output streams, 425-427
- filtering, 421, 427
- nonblocking I/O network connections, 492-499
- Path objects, 441-444
- reading, 420
- writing to, 421
- IOException, 195, 489, 536
 - file operations, 442
 - I/O streams, 422
- isAcceptable() method, 494
- isCancelled() method, 413
- isConnectible() method, 494
- isConnectionPending() method, 495
- isDone() method, 413
- isEditable() method, 268
- isEmpty() method, 241
- ISO-LATIN-1 character set, 487
- isReadable() method, 494
- isWritable() method, 494
- item events, 340, 349-351
- Item.java, 181
- ItemListener event listener, 340
- ItemProp.java, 658
- itemStateChanged() method, 349
- Iterator interface, 227-229, 235-238
- iterator() method, 235-238

J

- j2se tag, 403
- JAR files
 - creating as JNLP files, 396-404
 - signing, 658-659
- jarsigner, 658-659

- jar tag, 403
- jar utility, 650-652
- Java
 - applications, 136
 - compiling in Windows, 631-632
 - creating, 135-136
 - handling arguments, 138-139
 - passing arguments to, 137
 - running, 602-603, 631-632, 639
 - sample program, 629-631
 - speed, 642
 - case-sensitivity, 41
 - C++ versus, 11
 - development tools, selecting, 12-13, 616
 - documentation, 47
 - explained, 11
 - fonts, built-in, 371
 - history of, 10-11
- Java2D, 367
 - arcs, drawing, 382-383
 - casting objects, 85
 - color. *See* color
 - ellipses, drawing, 382
 - Graphics2D class, 368
 - coordinate system, 369-370
 - creating drawing surface, 368-369
 - lines
 - drawing, 377, 381
 - rendering attributes, 378-381
 - maps, drawing, 385-387
 - polygons
 - drawing, 377, 383-384
 - rendering attributes, 378-381
 - rectangles, drawing, 381

- text
 - antialiasing, 372
 - drawing, 370-372
 - finding font information, 372-375
 - user versus device
 - coordinate spaces, 378
- java21days website, 23
- Java API for XML Processing, 530
- java.awt.color package, 375
- java.awt.event package, 256, 340
 - ActionListener interface, 330, 460
 - adapter classes, 358, 456-457
 - event listeners, 455
- java.awt.geom package, 381
- java.awt package, 256
 - BorderLayout class, 322
 - CardLayout class, 326
 - Color class, 32, 375
 - Cursor class, 461
 - FlowLayout class, 315
 - Font class, 370
 - FontMetrics class, 373
- JavaBeans, 409
- java.beans package, 409
- javac compiler, 631, 641-642
- Java Class Library, 16, 159, 225, 278-281
- Java DB, 503
 - Access and MySQL
 - versus, 523
 - connecting to, 505-510
- JavaDB library, adding to projects, 512
- Java Development Kit. *See* JDK
- Javadoc comments, 47
- javadoc documentation tool, 646-650
- .java extensions, 641
- .java files, associating with text editor, 630
- /java folder (Android), 574
- JavaFX, 310
- java interpreter, 639-641
- java.io package, 279, 419. *See also* streams
 - File class, 441
 - IOException class, 195
 - PrintStream class, 79
- java.io.tmpdir system property, 657
- java.lang package, 32
 - exception classes, 195
 - Math class, 79
 - primitive type classes, 86
 - Runnable interface, 212
 - System class, 79, 431, 657
 - Thread class, 211
- Java Look and Feel Graphics Repository, 267
- java.net package, 469-470. *See also* networking
 - InetAddress class, 493
 - InetSocketAddress class, 493
 - URL class, 471
- java.nio.channels package, 484, 488
- java.nio.charset package, 484, 487
- java.nio.file package, 419
 - Files class, 442
 - Path objects, 441
- java.nio package, 469, 484. *See also* I/O (input/output) streams
 - buffers, 484-486
 - channels, 488-491
 - nonblocking I/O network connections, 492-499
- Java Plug-in, 393, 643
- Java Runtime Environment (JRE), installing, 392
- Java SE Development Kit 8, 12
- java.sql package, 505. *See also* JDBC (Java Database Connectivity)
 - DriverManager class, 509
- java.time package, 279
- java.util package, 159. *See also* data structures
 - StringTokenizer class, 69
 - TimeZone class, 657
- java.vendor system property, 657
- java -version command, 624
- java.version system property, 657
- Java Virtual Machine (JVM), 11, 392, 601, 623
- Java Web Start, 392-395
 - configuring web servers for, 405
 - description tag, 406
 - icon tag, 406-407
 - JNLP files, creating, 396-404
 - security, 405-406
- javax.swing package, 159, 256
 - BoxLayout class, 317
 - JButton class, 16
 - JComponent class, 264
 - JPanel class, 325, 368
 - SwingConstants interface, 309
 - SwingWorker class, 407
- javax.xml.parsers package, 530
- JButton class, 16, 261
- JCheckBox class, 272
- JCheckBox() constructor, 272
- JComboBox class, 274-275
- JComboBox() constructor, 275
- JComponent class, 256, 264
- JDBC (Java Database Connectivity), 504-505
 - data source connections
 - closing, 512
 - opening, 508-510
 - databases, accessing, 505
 - drivers, 505
- jdb debugger, 652-653
 - advanced commands, 655-656
 - applet debugging, 655
 - application debugging, 653-655

JDK (Java Development Kit),
12, 616, 637-638
 command line, 638-639
 configuring, 619
 command-line
 interface, 619-621
 creating folders,
 622-623
 opening folders,
 621-622
 running programs, 623
 setting CLASSPATH
 variable, 633-636
 setting PATH variable,
 624-628
 downloading, 638
 installing, 616-619
 system properties,
 656-658
 utilities
 appletviewer browser,
 642-646
 jar, 650-652
 jarsigner, 658-659
 javac compiler,
 641-642
 javadoc documentation
 tool, 646-650
 java interpreter,
 639-641
 jdb debugger, 652-656
 keytool, 658-659
 version number, 624

JDOM, 531

jEdit, 629

JFrame class, 257

JFrame() constructor, 257

JLabel class, 267

JLabel() methods, 267

JList class, 276

JList() constructor, 276

JMenuBar class, 303

JMenuBar() constructor, 305

JMenu class, 303

JMenuItem class, 303

JMenuItem() constructor, 303

JNLP files
 associating MIME
 types, 405
 creating, 396-404
 description tag, 406
 icon tag, 406-407
 security, 405-406

jnlp tag, 402

JOptionPane class, 286

JPanel class, 262, 325, 368

JPasswordField class, 268

JProgressBar class, 300

JProgressBar()
 constructor, 301

JPython language, 640

JRadioButton class, 272

JRE (Java Runtime
 Environment), installing, 392

IRuby language, 640

JScrollBar class, 297

JScrollPane class, 271, 296

JScrollPane() constructor,
 271, 296

JSlider class, 294

JSlider() constructor, 294

JTabbedPane class, 307

JTabbedPane()
 constructor, 307

JTextArea() constructors, 269

JTextComponent class, 268

JTextField class, 268

JTextField() constructor, 268

JToggleButton class, 272

JToolBar class, 297

JToolBar() constructor, 298

JudoScript language, 640

juncture styles (drawing
 strokes), 380

JUnit, 219

JVM (Java Virtual Machine),
 11, 392, 601, 623

K

Key class, 388

KeyAdapter class, 358, 457

keyboard events, 340,
 351-352

keyboard focus events, 340

KeyChecker.java, 358

KeyChecker2.java, 360

KeyListener event listener,
 340, 351

key-mapped data structures
 Dictionary class, 227
 HashMap class, 241-246
 Map interface, 240-241

keyPressed() method, 351

keyReleased() method, 351

keystores, 658

keytool, 658-659

keyTyped() method, 351

keywords. *See also*
 commands; statements
 abstract, 169
 break, 107, 119-120
 case, 107
 class, 126, 450
 continue, 119-120
 else, 104
 enum, 249
 extends, 126, 180
 final, 44, 167
 implements, 177, 212
 modifiers. *See* modifiers
 new, 454
 null, 97
 private, 159-161
 protected, 162
 public, 161
 return, 129
 static, 24, 74, 126-127,
 134, 164
 super, 150
 this, 130-131, 145, 327
 throws, 203-205

L

- labeled loops, 120
- labels, 255, 267
 - aligning, 267
 - creating, 267
 - menus, 304
 - progress bars, 301
 - sliders, 294-295
- lambda expressions, origin of term, 466. *See also* closures
- languages
 - JPython, 640
 - JRuby, 640
 - JudoScript, 640
 - NetRexx, 640
 - SQL (Structured Query Language), 504-505
- lastElement() method, 233
- last() method, 522
- layout managers, 314-315
 - alternatives to, 334
 - border layout, 322-324
 - box layout, 317-319
 - card layout, 325-333
 - combining, 324-325
 - creating, 314
 - flow layout, 315-317
 - grid layout, 320-321
 - insets, 333-334
- length instance variable, 99
- length() method, 76, 91
- less than or equal to
 - symbol ($\leq$) comparison operator, 57
- less than symbol ($<$)
 - comparison operator, 57
- lexical scope, 121
- LGPL (GNU Lesser General Public License), 531
- libraries
 - Apache Project, 279
 - for interfaces (GUI), 310
 - Java Class Library. *See* Java Class Library
 - XOM. *See* XOM
- licensing
 - Open Directory License, 559
 - for XOM, 531
- Line2D.Float class, 381
- line numbers in text editors, 629
- lines, drawing, 377
 - Line2D.Float class, 381
 - rendering attributes, 378-381
- lineTo() method, 384
- linked lists, 91
- linking
 - applets, URL objects, 471
 - node objects, 91
- list command (jdb), 654
- listeners. *See* event listeners
- listings
 - AllCapsDemo.java, 442
 - Alphabet.java, 316
 - AppInfo.html, 645
 - AppInfo.java, 644
 - AppInfo2.java, 647
 - ArrayCopier.java, 117
 - Authenticator.java, 269
 - Authenticator2.java, 272
 - Averager.java, 138
 - Border.java, 323
 - Box.java, 141
 - Box2.java, 146
 - BufferConverter.java, 490
 - BufferDemo.java, 429
 - Bunch.java, 320
 - ButtonFrame.java, 262
 - Buttons.java, 263
 - ByteReader.java, 424
 - ByteWriter.java, 426
 - Calculator.java, 347
 - ClosureMayhem.java, 464
 - CodeKeeper.java, 236
 - CodeKeeper2.java, 248
 - ComicBooks.java, 243
 - ComicBox.java, 452
 - ConsoleInput.java, 432
 - CursorMayhem.java, 462
 - CustomerReporter.java, 512
 - DayCounter.java, 108
 - DiceRoller.java, 410
 - DiceWorker.java, 408
 - DmozHandlerImpl.java, 563
 - DmozHandler.java, 562
 - DmozServer.java, 561
 - DomainEditor.java, 538
 - DomainWriter.java, 541
 - EqualsTester.java, 88
 - feed2.rss, 540
 - FeedBar.java, 299
 - FeedBar2.java, 305
 - FeedInfo.java, 291
 - feed.rss, 532
 - Finger.java, 477
 - FingerServer.java, 495
 - FormatChooser.java, 349
 - FormatFrame.java, 273
 - FormatFrame2.java, 275
 - GiftShop.java, 185
 - HalfDollars.java, 100
 - HalfLooper.java, 115
 - HelloUser.java, 630
 - HexReader.java, 200
 - HolidaySked.java, 231
 - IconFrame.java, 265
 - Info.java application, 292
 - InstanceCounter.java, 165
 - Item.java, 181
 - ItemProp.java, 658
 - KeyChecker.java, 358
 - KeyChecker2.java, 360
 - Map.java, 385
 - MarsApplication.java, 23
 - MarsRobot.java, 20
 - MousePrank.java, 353
 - NamedPoint class, 151
 - PageData.java, 396
 - PageData.jnlp, 400
 - Passer.java, 133
 - PointSetter.java, 73
 - PrimeFinder.java, 213
 - PrimeReader.java, 435

PrimeThreads.java, 215
 PrimeWriter.java, 434
 Printer.java, 148
 ProgressMonitor.java, 302
 ProgressMonitor2.java, 458
 QuoteData.java, 518
 RangeLister.java, 129
 RefTester.java, 80
 RssFilter.java, 543
 RssStarter.java, 534
 SantaActivity.java
 full text, 588
 starting text, 585
 SimpleFrame.java, 260
 SiteClient.java, 558
 Slider.java, 295
 SourceReader.java, 439
 Spartacus.java, 601
 Stacker.java, 318
 Storefront.java, 184
 StringChecker.java, 76
 Subscriptions.java, 277
 SurveyFrame.java, 333
 SurveyWizard.java, 331
 TabPanels.java, 308
 TextFrame.java, 373
 TimeServer.java, 481
 TitleBar.java, 343
 TokenTester.java, 69
 Variables.java, 45
 Weather.java, 53
 WebReader.java, 471
 workbench.rss, 526
 XML-RPC request, 552
 XML-RPC response, 553
 lists, 276-278
 literals, 47
 Boolean, 49
 character, 49-50
 integer, 47-49
 string, 50-51
 load factor (hash maps), 241
 loading
 classes, 508
 database drivers, 508
 locals command (jdb), 654

local scope, 131
 local variables, 39
 locations, frames, 258
 logical fonts, 371
 logical operators, 57-58
 long data type, 42, 83
 look and feel, 256
 looping through data
 structures, 235-238
 loops
 breaking, 119
 do, 118-119
 for, 113-116
 increments, 113
 index values, 114
 initialization, 113
 labeling, 120
 restarting, 119-120
 run() method, stopping
 threads, 217-218
 tests, 113
 while, 116-118

M

main-class attribute, 403
 main classes, designating, 136
 main() method, 24,
 135-136, 639
 importance of, 603
 as public, 161
 MalformedURLException,
 195, 471
 managing
 errors. *See* error-handling
 exceptions. *See* error-
 handling
 memory, 71-72
 manifest files, configuring in
 Android Studio, 581
 /manifests/AndroidManifest.
 xml, 574
 Map interface, 240-241
 Map.java, 385
 maps, drawing, 385-387
 MarsApplication.java, 23
 MarsRobot.java, 20
 Math class, 79, 280
 math operators. *See*
 arithmetic operators
 MD command (MS-DOS), 622
 member variables. *See*
 instance variables
 memory
 allocating, 71
 deallocating, 71
 managing, 71-72
 reclaiming, 72
 memory command (jdb), 656
 menu commands,
 appletviewer browser,
 643-644
 menus, 303-307
 creating, 304
 labels, 304
 separators, adding, 304
 message dialog boxes, 289
 MessageDialog class,
 286, 289
 methods, 18
 abstract methods, 169
 accept(), 479
 access control, 159
 accessor methods,
 164, 187
 comparison of
 types, 163
 default access, 159
 inheritance, 163
 private access, 159-161
 protected access, 162
 public access, 161
 actionPerformed()
 action events, 342, 345
 buttons, 330
 adapter classes, 357-359
 add()
 array lists, 233-234
 border layouts, 323
 card layouts, 326
 check boxes/radio
 buttons, 273
 containers, 262
 menus, 304

- addActionListener(), 330, 341, 345
- addAttribute(), 533
- addFocusListener(), 341
- addHandler(), 560
- addItem(), 275
- addItemListener(), 341
- addMouseListener(), 341
- addMouseMotionListener(), 341
- addSeparator(), 304
- addTab()panes, 307
- addTextListener(), 341
- addWindowListener(), 341
- afterLast(), 521
- allocate(), 485
- append(), 269
- appendChild(), 533
- beforeFirst(), 521
- build(), 536
- calling, 18, 75-79
- chaining, 655
- channel(), 495
- charAt(), 77, 91
- charWidth(), 373
- in class hierarchy, 30
- class methods, 19, 79-80, 134-135
 - accessing, 165
 - calling, 80
 - defining, 134
- clear()
 - array lists, 235
 - hash maps, 242
- close()
 - buffered character streams, 441
 - character streams, 440
 - client-side sockets, 476
 - data source
 - connections, 512
 - data streams, 434
 - file output
 - streams, 425
 - streams, 420-421
- closePath(), 384
- configureBlocking(), 493
- connect(), 493
- constructors, 69, 144-145
 - calling, 144, 151
 - calling from another constructor, 145-146
 - definition of, 71
 - naming, 144
 - overloading, 146-147
 - overriding, 150-152
- contains(), 235
- containsKey(), 242
- containsValue(), 242
- createFont(), 371
- createStatement(), 510, 522
- decode(), 488
- defining, 21, 128-130
 - this keyword, 130-131
 - variable scope, 131-132
- delete(), 442
- deprecated methods, 642
- doinBackground(), 408
- draw(), 384
- drawString(), 370
- empty(), 239
- encode(), 488
- equals(), 88, 242
- event-handling methods, 342-345
- execute(), 557
- executeQuery(), 510-511
- fill(), 384
- final abstract methods, 187
- final methods, 167-168
- finishConnect(), 495
- first(), 521
- flip(), 485
- floor(), 280
- flush(), 428
- focusGained(), 346
- focusLost(), 346
- forName()
 - character sets, 487
 - database drivers, 508
- get()
 - array lists, 233
 - buffers, 484
 - elements, 537
 - Map interface, 241
- getActionCommand(), 345
- getAppletInfo(), 644-645, 655
- getChannel(), 488
- getChar(), 486
- getChild(), 537
- getChildElements(), 537
- getClass(), 90
- getClickCount(), 352
- getColor(), 377
- getConnection(), 509
- getContentType(), 474
- getDate(), 511
- getDefault(), 441
- getDouble()
 - byte buffers, 486
 - database records, 511
- getErrorCode(), 511
- getFirstChildElement(), 537
- getFloat()
 - byte buffers, 486
 - database records, 511
- getFontMetrics(), 373
- getHeaderField(), 474
- getHeaderFieldKey(), 474
- getHeight(), 373
- getIcon(), 267
- getId(), 586
- getInsets(), 334
- getInt()
 - byte buffers, 486
 - database records, 511
- getItem(), 349
- getItemAt(), 275
- getItemCount(), 275
- getKeyChar(), 351
- getLong()
 - byte buffers, 486
 - database records, 511
- getMessage(), 197

- getParameterInfo(), 644-645
- getPath(), 441
- getPoint(), 352
- getProperties(), 657
- getProperty(), 657
- getResponseCode(), 474
- getResponseMessage(), 474
- getRootElement(), 536
- getSelectedIndex(), 275
- getSelectedItem(), 275
- getSelectedText(), 268
- getSelectedValuesList(), 277
- getShort(), 486
- getSize(), 264
- getSource(), 342, 345
- getSQLState(), 511
- getStateChange(), 349
- getString(), 511
- getText(), 267-268
- getX(), 352
- getXmlRpcServer(), 559
- getY(), 352
- grouping, 79
- hashCode(), 242
- hasNext(), 228
- indexOf(), 77, 235
- insert(), 269
- insertChild(), 541
- instance methods, 135
- in interfaces, 31, 179-180
- isAcceptable(), 494
- isCancelled(), 413
- isConnectible(), 494
- isConnectionPending(), 495
- isDone(), 413
- isEditable(), 268
- isEmpty(), 241
- isReadable(), 494
- isWritable(), 494
- itemStateChanged(), 349
- iterator(), 235-238
- JLabel(), 267
- keyPressed(), 351
- keyReleased(), 351
- keyTyped(), 351
- last(), 522
- lastElement(), 233
- length(), 76, 91
- lineTo(), 384
- main(), 24, 135-136, 639
 - importance of, 603
 - as public, 161
- mouseClicked(), 352, 362
- mouseDragged(), 353
- mouseEntered(), 352
- mouseExited(), 352
- mouseMoved(), 353
- mousePressed(), 352
- mouseReleased(), 352
- move(), 442
- moveTo(), 384
- newDecoder(), 488
- newEncoder(), 488
- newLine(), 441
- next()
 - Iterator interface, 228
 - resultsets, 521
 - socket channels, 494
- onCreate(), 586
- open(), 493
- overloading, 128
 - advantages, 139
 - creating overloaded methods, 140-143
 - definition of, 139
 - troubleshooting, 140
- overriding, 30-31, 147-149
 - advantages, 149-150
 - super keyword, 150
- pack(), 258, 310
- paintComponent(), 368, 377
- parameter lists, 128
- parseInt(), 86
- passing arguments to, 132-134
- peek(), 239
- pop(), 239
- position(), 485
- prepareStatement(), 514
- previous(), 522
- print(), 46
- println(), 46, 79
- printStackTrace(), 199
- private abstract methods, 187
- private methods as final, 168
- processClicks(), 586
- protecting, 170
- push(), 239
- put()
 - buffers, 485
 - Map interface, 240
- putChar(), 486
- putDouble(), 486
- putFloat(), 486
- putInt(), 486
- putLong(), 486
- putShort(), 486
- random(), 280
- read()
 - buffered character streams, 438
 - buffered input streams, 428
 - byte buffers, 489
 - character streams, 437
 - file input streams, 423
 - filters, 421
 - streams, 420
- readBoolean(), 433
- readByte(), 433
- readDouble(), 433
- readFloat(), 433
- readInt(), 433
- readLine(), 438
- readLong(), 433
- readShort(), 433
- readUnsignedByte(), 433
- readUnsignedShort(), 433
- register(), 493
- remove()
 - array lists, 234-235
 - Map interface, 241
 - socket channels, 495

- removeChild(), 538
- requestFocus(), 346
- return types, 128-129
- run(), 213, 217-218
- search(), 239
- select(), 494
- selectedKeys(), 494
- set(), 234
- setActionCommand(), 346
- setAsciiStream(), 515
- setBackground(), 377
- setBinaryStream(), 515
- setBoolean(), 515
- setBounds()
 - components, 335
 - frames, 258
- setByte(), 515
- setBytes(), 515
- setCharacterStream(), 515
- setColor(), 376
- setConfig(), 556
- setContentView(), 586
- setCursor(), 461
- setDate(), 515
- setDefaultCloseOperation(), 259
- setDouble(), 515
- setEchoChar(), 268
- setEditable()
 - combo boxes, 275
 - text fields, 268
- setEnabled(), 264
- setFloat(), 515
- setFollowRedirects(), 474
- setFont(), 281, 371
- setHgap(), 320
- setIcon(), 267
- setIndentation(), 541
- setInt(), 515
- setJMenuBar(), 305
- setLayout()
 - card layouts, 326
 - containers, 314
 - panels, 325
- setLineWrap(), 269
- setListData(), 277
- setLong(), 515
- setLookAndFeel()
 - dialog boxes, 293
 - GUIs, 259
- setMajorTickSpacing(), 294
- setMaximum(), 301
- setMinimum(), 301
- setMinorTickSpacing(), 294
- setNull(), 516
- setPaint(), 378
- setPaintLabels(), 295
- setPaintTicks(), 295
- setPreferredSize(), 296
- setRenderingHint(), 372
- setSelected(), 272
- setSelectedIndex(), 275
- setServerURL(), 556
- setShort(), 515
- setSize()
 - components, 264
 - frames, 257
- setSoTimeout(), 475
- setString(), 515
- setStringPainted(), 301
- setStroke(), 380
- setText(), 267-268
- setValue(), 301
- setVgap(), 320
- setVisible()
 - components, 264
 - frames, 258
- setVisibleRowCount(), 277
- setWrapStyleWord(), 269
- show(), 326
- showConfirmDialog(), 287
- showInputDialog(), 288
- showMessageDialog(), 289
- showOptionDialog(), 290
- signatures, 128, 139
- size()
 - array lists, 235
 - elements, 537
 - file channels, 489
 - Map interface, 241
- start(), 212
- startActivity(), 587
- static, 164-167
- stringWidth(), 373
- substring(), 77, 201
- super(), 151
- in superclasses,
 - calling, 150
- System.out.format(), 77-78
- throwing exceptions,
 - 202, 207
 - checked, 204
 - inheritance, 206
 - nested handlers, 208-209
 - throws clause, 203
 - unchecked, 204
- toFile(), 441
- toPath(), 442
- toUpperCase(), 77
- toXML(), 534
- trimToSize(), 235
- valueOf(), 79
- windowActivated(), 357
- windowClosed(), 357
- windowClosing(), 357
- windowDeactivated(), 357
- windowDeiconified(), 357
- windowIconified, 357
- windowOpened(), 357
- wrap(), 484
- write()
 - buffered character streams, 441
 - buffered output streams, 428
 - character streams, 440
 - char versus int data types, 445
 - file output streams, 425
 - filters, 421
 - streams, 421
 - XML documents, 541
- writeBoolean(), 433
- writeByte(), 433
- writeDouble(), 433
- writeFloat(), 433
- writeInt(), 433

- writeLong(), 433
 - writeShort(), 433
 - in XML-RPC, 552
 - zero-based, 77
 - methods command (jdb), 656
 - Microsoft Word, 629
 - MIME types, associating, 405
 - minus equal (=) assignment operator, 54
 - minus sign (-)
 - decrement operator (--), 55-56
 - negative numbers, 48
 - subtraction operator, 52
 - modifiers, 158
 - abstract, 169
 - access control, 159
 - accessor methods, 164
 - comparison of types, 163
 - default access, 159
 - inheritance, 163
 - private access, 159-161
 - protected access, 162
 - public access, 161
 - final, 167
 - multiple, 158
 - private, 159-161
 - protected, 162
 - public, 161
 - return types versus, 158
 - static, 164
 - modifying. *See also* changing; editing
 - class variable values, 75
 - functions, 638
 - instance variable values, 73-74
 - operator precedence, 60
 - superclasses, 27
 - XML documents, 536-540
 - modulus operator, 52
 - MouseAdapter class, 358, 457
 - mouseClicked() method, 352, 362
 - mouseDragged() method, 353
 - mouseEntered() method, 352
 - MouseEvent objects, 352
 - mouse events, 340, 352, 362
 - mouseExited() method, 352
 - MouseListener, 340, 352
 - MouseMotionAdapter class, 358
 - MouseMotionListener, 340, 352
 - mouseMoved() method, 353
 - mouse movement events, 340, 352-357
 - MousePrank.java, 353
 - mousePressed() method, 352
 - mouseReleased() method, 352
 - move() method, 442
 - moveTo() method, 384
 - MS-DOS, 620
 - CLASSPATH variable
 - Windows 7-10, 633-635
 - Windows 98/Me, 635-636
 - commands, 621-622
 - folders
 - creating, 622-623
 - opening, 621-622
 - PATH variable
 - Windows 7-10, 625-627
 - Windows 98/Me, 627-628
 - programs, running, 623
 - prompt. *See* command line
 - multidimensional arrays, 102
 - multiline comments, 46
 - multiple bytes, writing, 425
 - multiple files, compiling, 641
 - multiple inheritance, 31, 176
 - multiple interfaces, implementing, 177-178
 - multiple modifiers, 158
 - multiplication operator, 52
 - multitasking. *See* threads
 - MySQL databases, Java DB versus, 523
- ## N
- NamedPoint class, 151
 - naming
 - class variables, 127
 - conflicts, reducing, 170-172
 - constants, 44
 - methods, constructors, 144
 - packages, 173-174
 - resources, 580
 - variables, 40-41
 - navigating
 - database records, 521-522
 - records, 511
 - negative numbers, as literals, 48
 - nesting
 - exception handlers, 208-209
 - if statements, 106
 - method calls, 78-79
 - XML tags, 527
 - NetBeans, 12, 597
 - adding
 - Apache XML-RPC to, 555
 - JavaDB library to projects, 512
 - XOM to, 532
 - connecting to databases, 505-508
 - database connection information, viewing, 510
 - database tables, creating, 517
 - grouping packages, 183
 - installing, 598
 - Java applications, running, 602-603
 - Java classes
 - compiling, 601
 - creating, 600-602
 - importing, 173
 - main classes, designating, 136
 - organizing graphics, 266

- panes, expanding/shrinking, 604
- projects, creating, 19, 598-600
- resources for
 - information, 605
- Run File versus Run Project commands, 134
- setting command-line arguments, 109
- troubleshooting in, 603
- updating, 598
- NetRexx language, 640
- networking, 470
 - nonblocking I/O
 - connections, 492-499
 - security, 499
 - sockets, 475-479
 - client-side, 475-476
 - server-side, 479-483
 - streams, 470-475
 - web services, 549
 - XML-RPC. *See* XML-RPC
- newDecoder() method, 488
- newEncoder() method, 488
- new keyword, 454
- newLine() method, 441
- new operator, 59, 68
 - creating objects with, 68-70
 - instantiating arrays, 97
- next() method
 - Iterator interface, 228
 - resultsets, 521
 - socket channels, 494
- NoClassDef error, 633-635
- Node class, 533
- node objects, linking, 91
- nodes, adding children to parents, 533
- nonblocking I/O network connections, 492-499
- nonexclusive check boxes, 273
- non-runtime exceptions, 219
- NoSuchFileException, 442
- NotePad, 628

- not equal symbol (!=)
 - comparison operator, 57, 88
- NOT operator, 58
- null keyword, 97
- NullPointerException, 194
- number literals, 47-49
- numbers
 - binary, 48
 - converting strings to, 86
 - floating-point, 42
 - formatting display, 77-78
 - hexadecimal, 49
 - integers, 42
 - octal, 48
- nu.xom.canonical package, 542
- nu.xom.converters package, 542
- nu.xom package, 533
- nu.xom.xinclude package, 542
- nu.xom.xslt package, 542

O

- Object class, 26
- object-oriented programming (OOP), 11-14. *See also* classes; objects
- objects, 14. *See also* instances
 - ArrayList, 556
 - arrays, creating, 97-98
 - attributes, 17-18
 - in class hierarchies, 29
 - defining, 17
 - behavior, 18-19
 - ButtonGroup, 273
 - ByteBuffer, 489
 - casting, 82-85, 565
 - classes and, 14-16
 - Color, creating, 376
 - comparing, 87-89
 - creating, 68
 - arguments, 68
 - with closures, 460-465
 - with constructors, 71
 - with new operator, 68-70
 - StringTokenizer objects, 69-70
- current, referring to, 130
- determining class of, 89-90
- Dimension, 264, 296
- Document, 533
- drawing, 384
- Element, creating, 533
- encapsulating, 161
- File, creating, 441
- FileChannel, 488
- Font, creating, 370-372
- GeneralPath, creating, 384
- ImageIcon, 265
- inheritance, 29-31
- initializing, 71
- memory, allocating/deallocating, 71
- MouseEvent, 352
- nodes, linking, 91
- Path, 441-444
- primitive types, 91
 - converting, 86-87
- Rectangle, frame boundaries, 258
- references, 80-82
- ResultSet, 511
- reusing, 15-16
- Selector, 493
- Serializer, 540
- Set, 494
- String, creating, 21
- URL, creating, 471
- object variables. *See* instance variables
- object wrappers, 86
- obscuring password fields, 268
- octal numbers, 48
- offline-allowed tag, 401
- onCreate() method, 586
- online storefronts, creating, 181-187
- OOP (object-oriented programming), 11-14. *See also* classes; objects

- Open Directory License, 559
- Open Directory Project, 559
- opening
 - data source connections, 508-510
 - folders in MS-DOS, 621-622
 - socket connections, 475
 - sockets, 493
 - streams over Internet, 470-475
- opening tags (XML), 401, 527
- open() method, 493
- operators, 52
 - arithmetic, 52-54
 - assignment, 40, 43, 54-55
 - bitwise, 59
 - comparison, 56-57, 88
 - concatenation, 82
 - decrement ($-$), 55-56
 - diamond operator, 247
 - increment ($++$), 55-56
 - instanceof, 59, 90, 343
 - list of, 61
 - logical, 57-58
 - new, 59, 68
 - creating objects with, 68-70
 - instantiating arrays, 97
 - postfix, 55-56
 - precedence, 58-60
 - prefix, 55-56
 - string concatenation, 60-61
 - ternary, 59, 112
- option dialog boxes, 290-291
- AlertDialog class, 286, 290-291
- options (commands), 638-639
- order of precedence, 58-60
- organizing
 - classes, 25, 158, 170
 - creating hierarchies, 27-29
 - inheritance, 25-31
 - interfaces, 31-32
 - packages, 32, 45

- graphics in NetBeans, 266
- projects in Android Studio, 574-575
- org.apache.xmlrpc
 - package, 554
- org.apache.xmlrpc.client
 - package, 556
- org.apache.xmlrpc.server
 - package, 559
- org.apache.xmlrpc.webserver
 - package, 559
- orientation
 - progress bars, 301
 - sliders, 294
 - toolbars, 297
- OR operators, 58
- os.name system property, 657
- os.version system property, 657
- OutputStream class, 422
- output streams, 420. *See also* streams
 - buffered output streams, 428
 - creating, 420
 - data output streams, 433
 - file output streams, 425-427
- OutputStreamWriter class, 440
- overflow (variable assignment), 62
- overloading
 - constructors, 146-147
 - methods, 128
 - advantages, 139
 - creating, 140-143
 - definition of, 139
 - troubleshooting, 140
- overriding
 - constructors, 150-152
 - methods, 30-31, 147-150
 - scrollbars, 297

P

- package declaration, 175
- package statement, 32, 63

- packages, 32, 45, 169
 - access control, 175
 - advantages, 170
 - android.content, 587
 - android.support.v7.app, 586
 - creating, 173-175, 640
 - grouping in NetBeans, 183
 - importing, 171
- java.awt, 256
 - BorderLayout
 - class, 322
 - CardLayout class, 326
 - Color class, 32, 375
 - Cursor class, 461
 - FlowLayout class, 315
 - Font class, 370
 - FontMetrics class, 373
- java.awt.color, 375
- java.awt.event, 256, 340
 - ActionListener interface, 330, 460
 - adapter classes, 358, 456-457
 - event listeners, 455
- java.awt.geom, 381
- java.beans, 409
- java.io, 279, 419
 - File class, 441
 - IOException class, 195
 - PrintStream class, 79
- java.lang, 32
 - exception classes, 195
 - Math class, 79
 - primitive type classes, 86
 - Runnable interface, 212
 - System class, 79, 431, 657
 - Thread class, 211
- java.net, 469-470
 - InetAddress class, 493
 - InetSocketAddress class, 493
 - URL class, 471

- java.nio, 469, 484
 - buffers, 484-486
 - channels, 488-491
 - nonblocking I/O
 - network connections, 492-499
- java.nio.channels, 484, 488
- java.nio.charset, 484, 487
- java.nio.file, 419, 441-442
- java.sql, 505, 509
- java.time, 279
- java.util, 159. *See also* data structures
 - StringTokenizer class, 69
 - TimeZone class, 657
- javax.swing, 159, 256
 - BoxLayout class, 317
 - JButton class, 16
 - JComponent class, 264
 - JPanel, 368
 - JPanel class, 325
 - SwingConstants interface, 309
 - SwingWorker class, 407
- javax.xml.parsers, 530
- nu.xom, 533
- nu.xom.canonical, 542
- nu.xom.converters, 542
- nu.xom.xinclude, 542
- nu.xom.xslt, 542
- org.apache.xmlrpc, 554
- org.apache.xmlrpc.client, 556
- org.apache.xmlrpc.server, 559
- org.apache.xmlrpc.webserver, 559
- referencing, 170
- pack() method, 258, 310
- PageData.java, 396
- PageData.jnlp, 400
- paintComponent() method, 368, 377
- Paint interface, 378
- panels, 262, 325
 - absolute component placement, 334
 - card layouts, 326
 - components, adding, 325
 - creating, 325, 368
 - insets, 333-334
 - scrolling panes, 255, 271-272, 296-297
 - tabbed panes, 307-310
- panes (NetBeans), expanding/shrinking, 604
- @param tag (javadoc), 650
- parameter lists (methods), 128
- parameters (XML-RPC), 553
- parentheses ()
 - arguments, 68
 - grouping expressions, 59-60
- parent nodes, adding child nodes to, 533
- ParseException errors, 536
- parseInt() method, 86
- Passer.java, 133
- passing
 - arguments
 - to applications, 137
 - to methods, 132-134
 - exceptions, 204-205
- password fields, obscuring, 268
- Path objects, 441-444
- paths, relative, 444
- PATH variable (MS-DOS)
 - Windows 7-10, 625-627
 - Windows 98/Me, 627-628
- peek() method, 239
- percent sign (%) modulus operator, 52
- performance
 - improving, 407-412
 - Java programs, 642
- period (.)
 - accessing methods and variables, 59
 - dot notation, 73
- permalinks, 527
- pipe character (|) OR operator, 58
- platform neutrality, 11
- plus equal (+=) assignment operator, 54
- plus sign (+)
 - addition operator, 52
 - concatenation operator, 82
 - increment operator (++), 55-56
 - string concatenation, 60-61
- pointers (C/C++), 82, 91. *See also* arrays; references
- PointSetter.java, 73
- polygons, drawing, 377, 383-384
 - arcs, 382-383
 - ellipses, 382
 - rectangles, 381
 - rendering attributes, 378-381
- pop() method, 239
- port numbers, selecting, 480
- position() method, 485
- postfix operators, 55-56
- post requests, 551
- precedence of operators, 58-60
- prefix operators, 55-56
- prepared statements, 514-516
- PreparedStatement class, 514
- prepareStatement() method, 514
- preparing resources in Android Studio, 579-580
- previous() method, 522
- PrimeFinder.java, 213
- PrimeReader.java, 435
- PrimeThreads.java, 215
- PrimeWriter.java, 434
- primitive types, 42-43
 - casting, 82-84
 - objects, 91
 - converting, 86-87
- print command (jdb), 654

Printer.java, 148
 println() method, 46, 79
 print() method, 46
 printStackTrace() method, 199
 PrintStream class, 79
 private abstract methods, 187
 private access, 159-161
 private methods, as final, 168
 private modifier, 158-161
 problem-solving. *See* troubleshooting
 procedural programming, 13
 procedures (XML-RPC), 553
 processClicks() method, 586
 processing XML
 with Java, 530
 with XOM, 530-532
 creating XML documents, 532-535
 evaluating XOM, 542-545
 formatting XML documents, 540-542
 modifying XML documents, 536-540
 programming
 object-oriented, 11-14
 procedural, 13
 programs
 classes versus, 125
 Java. *See* Java applications
 MS-DOS, running, 623
 readability, improving, 46
 running, 22-25
 progress bars, 300-303
 labels, 301
 orientation, 301
 updating, 301
 ProgressMonitor.java, 302
 ProgressMonitor2.java, 458
 projects
 adding JavaDB library to, 512
 closing in Android Studio, 579

 creating, 19
 in Android Studio, 572-574, 579
 in NetBeans, 598-600
 organizing in Android Studio, 574-575
 properties, system, 656-658
 property change listeners, 409
 PropertyHandlerMapping class, 560
 protected access, 162
 protected modifier, 158, 162
 protecting classes/methods/variables, 170. *See also* access control
 public access, 161, 175
 public modifier, 158, 161
 push() method, 239
 putChar() method, 486
 putDouble() method, 486
 putFloat() method, 486
 putInt() method, 486
 putLong() method, 486
 put() method
 buffers, 485
 Map interface, 240
 putShort() method, 486

Q

queries, 504, 508-511
 question mark (?) in SQL statements, 514
 quitting. *See* stopping
 quotation marks (") in arguments, 137
 QuoteData.java, 518

R

radio buttons, 255, 272-274
 event handling
 action events, 345-346
 item events, 349-351
 exclusive, 273
 random() method, 280

RangeLister.java, 129
 RDF Site Summary, 528
 readability, improving
 expressions, 60
 programs, 46
 readBoolean() method, 433
 readByte() method, 433
 readDouble() method, 433
 Reader class, 437
 readFloat() method, 433
 reading
 buffered character streams, 438
 buffered input streams, 428
 C programs, 444
 database records, 508-513
 data input streams, 433
 streams, 420
 text files, 437-440
 readInt() method, 433
 readLine() method, 438
 readLong() method, 433
 read() method
 buffered character streams, 438
 buffered input streams, 428
 byte buffers, 489
 character streams, 437
 file input streams, 423
 filters, 421
 streams, 420
 readShort() method, 433
 readUnsignedByte() method, 433
 readUnsignedShort() method, 433
 Really Simple Syndication. *See* RSS
 reclaiming memory, 72
 records
 in databases
 navigating, 521-522
 reading, 508-513
 writing, 514-521
 navigating, 511

Rectangle2D.Float class, 382

Rectangle objects, frame boundaries, 258

rectangles, drawing, 381

reducing name conflicts, 170

references, 80-82

- arrays, 99
- to current objects, 130
- passing arguments by, 132
- to packages, 170

RefTester.java, 80

register() method, 493

relative paths, 444

Reload command (appletviewer), 643

remote method invocation (RMI), 550

remote procedure calls (RPC), 549-550. *See also* XML-RPC

removeChild() method, 538

remove() method

- array lists, 234-235
- Map interface, 241
- socket channels, 495

removing stack elements, 239

renaming files, 442

rendering attributes (Java2D), 378-381

RenderingHint.Key class, 388

requestFocus() method, 346

requests (XML-RPC)

- responding to, 553-554
- sending, 551-552

requirements for HAXM, 612

/res folder (Android), 574, 579

/res/layout folder (Android), 582

/res/mipmap folder (Android), 579

reserved words. *See* keywords; modifiers

resizing

- components, 264
- frames, 310
- NetBeans panes, 604
- scrolling panes, 296

resources

- for information
 - author contact information, 608
 - book website, 607
 - NetBeans, 605
 - naming, 580
 - preparing in Android Studio, 579-580
- resources tag, 402
- responding to XML-RPC requests, 553-554
- Restart command (appletviewer), 643
- restarting loops, 119-120
- ResultSet object, 511
- resultsets, navigating, 521-522
- return keyword, 129
- @return tag (javadoc), 647
- return types, 128
 - methods, void, 129
 - modifiers versus, 158
- return values, 38, 52
- reusing objects, 15-16
- R.java class, 586
- RMI (remote method invocation), 550
- RPC (remote procedure calls), 549-550. *See also* XML-RPC
- RSS (Really Simple Syndication), 525, 528
 - evaluating XOM, 542-545
 - versions 1.0 and 2.0, 546
 - well-formed XML, 528
 - XML and, 526
 - XML documents
 - creating, 532-535
 - formatting, 540-542
 - modifying, 536-540
- RSS Advisory Board, 528
- RssFilter.java, 543
- RssStarter.java, 534
- run command (jdb), 654
- Run File command, Run Project command versus, 134
- run() method, 213, 217-218

Runnable interface, 191, 212

running

- Android apps, 577-578, 589-591
- applications, 639
- bytecode, 639
- Java applications, 602-603
 - in Windows, 631-632
- JVM, 623
- programs, 22-25
 - in MS-DOS, 623
- self-signed JAR files, 659
- telnet, 482
- threads, 213

Run Project command, Run File command versus, 134

runtime errors, compiler errors versus, 247

RuntimeException class, 195

runtime exceptions, 194, 204, 219

S

Sams Teach Yourself Android Application Development in 24 Hours (Delessio, Darcey, Conder), 589

Sams Teach Yourself SQL in 24 Hours (Stephens, Jones, Plew), 510

SantaActivity.java

- full text, 588
- starting text, 585

saving source files, 630

SAX (Simple API for XML), 530

scope

- inner classes, 450
- lexical scope, 121
- variables, 103, 131-132

scrollbars, 296

- configuring, 271
- overriding, 297

scrolling

- panes, 255, 271-272, 296-297
- tabbed panes, 307

- ScrollPaneConstants interface, 271, 297
- searching stacks, 239
- search() method, 239
- security
 - digital signatures, 404
 - Java DB, 506
 - Java Web Start applications, 394-395
 - networking, 499
- SecurityException, 442
- security tag, 405-406
- @see tag (javadoc), 650
- selectedKeys() method, 494
- selecting
 - development tools, 12-13, 616
 - font styles, 371
 - port numbers, 480
 - substrings, 201
 - text editors, 628-629
- SelectionKey class, 494
- select() method, 494
- Selector object, 493
- self-signed JAR files, running, 659
- semicolon (;), statement termination character, 38
- sending XML-RPC requests, 551-552
- separators (menus), adding, 304
- @serial tag (javadoc), 647
- Serializer class, 540
- Serializer() constructor, 541
- servers, XML-RPC, 559-564
- server-side sockets, 479-480
 - designing server applications, 480-482
 - nonblocking servers, 493-499
 - testing server applications, 482-483
- ServerSocket class, 479
- setActionCommand() method, 346
- setAsciiStream() method, 515
- setBackground() method, 377
- setBinaryStream() method, 515
- setBoolean() method, 515
- setBounds() method
 - components, 335
 - frames, 258
- setByte() method, 515
- setBytes() method, 515
- setCharacterStream() method, 515
- setColor() method, 376
- setConfig() method, 556
- setContentView() method, 586
- setCursor() method, 461
- setDate() method, 515
- setDefaultCloseOperation() method, 259
- setDouble() method, 515
- setEchoChar() method, 268
- setEditable() method
 - combo boxes, 275
 - text fields, 268
- setEnabled() method, 264
- setFloat() method, 515
- setFollowRedirects() method, 474
- setFont() method, 281, 371
- setHgap() method, 320
- setIcon() method, 267
- setIndentation() method, 541
- setInt() method, 515
- setJMenuBar() method, 305
- setLayout() method
 - card layouts, 326
 - containers, 314
 - panels, 325
- setLineWrap() method, 269
- setListData() method, 277
- setLong() method, 515
- setLookAndFeel() method
 - dialog boxes, 293
 - GUIs, 259
- setMajorTickSpacing() method, 294
- setMaximum() method, 301
- set() method, 234
- setMinimum() method, 301
- setMinorTickSpacing() method, 294
- setNull() method, 516
- Set object, 494
- setPaintLabels() method, 295
- setPaint() method, 378
- setPaintTicks() method, 295
- setPreferredSize() method, 296
- setRenderingHint() method, 372
- setSelected() method, 272
- setSelecteIndex() method, 275
- setServerURL() method, 556
- setShort() method, 515
- setSize() method
 - components, 264
 - frames, 257
- setSoTimeout() method, 475
- setString() method, 515
- setStringPainted() method, 301
- setStroke() method, 380
- setText() method, 267-268
- setting
 - background color, 377
 - breakpoints, 653-655
 - drawing colors, 376-377
 - system properties, 657
- setValue() method, 301
- setVgap() method, 320
- setVisible() method
 - components, 264
 - frames, 258
- setVisibleRowCount() method, 277
- setWrapStyleWord() method, 269
- shared behavior, 31-32
- shared values, defining, 43
- shell prompt. See command line
- short data type, 42
- showConfirmDialog() method, 287
- showInputDialog() method, 288

- showMessageDialog()
 - method, 289
- show() method, 326
- showOptionDialog()
 - method, 290
- shrinking NetBeans panes, 604
- signatures (digital), 404
- signatures (methods), 128, 139
- signing code, 658-659
- Simple API for XML (SAX), 530
- SimpleFrame.java, 260
- Simple Object Access Protocol (SOAP), 551
- @since tag (javadoc), 650
- single inheritance, 31, 176
- single-step execution, 652
- SiteClient.java, 558
- size() method
 - array lists, 235
 - elements, 537
 - file channels, 489
 - Map interface, 241
- sizing
 - components, 264
 - frames, 257, 310
 - scrolling panes, 296
- slash character (/), XML tags, 401
- Slider.java, 295
- sliders, 294-296
 - advantages, 294
 - labels, 294-295
 - orientation, 294
- SOAP (Simple Object Access Protocol), 551
- SocketChannel class, 493
- Socket class, 475
- SocketImpl class, 480
- SocketImplFactory
 - interface, 480
- sockets, 475-479
 - client-side
 - closing, 476
 - nonblocking clients, 493-499
 - opening, 475
 - opening, 493
 - server-side, 479-480
 - designing server applications, 480-482
 - nonblocking servers, 493-499
 - testing server applications, 482-483
 - timeout values, 475
- solving problems. *See* troubleshooting
- source code. *See also* listings
 - comments in, 646
 - converting, 641
 - writing in Android Studio, 584-591
- source files
 - creating, 629
 - saving, 630
- SourceReader.java, 439
- sources (casting), 83
- Spartacus.java, 601
- specifying class files, 640
- speed of Java programs, 642
- splash screens, 407
- SQLException, 509-511
- SQL (Structured Query Language), 504-505
 - prepared statements, 514-516
 - queries, 508-511
- square brackets ([]), arrays, 59, 96
- sRGB color system, 375
- Stack class, 227, 238-239
- Stacker.java, 318
- stack frames, 656
- stacks, 227, 238-239, 656
- Standard Widget Toolkit (SWT), 310
- startActivity() method, 587
- Start command
 - (appletviewer), 643
- start() method, 212
- Statement interface, 510
- statements, 38. *See also* commands; keywords; modifiers
 - block statements, 38, 103, 121
 - conditionals
 - if, 104-105
 - nested if, 106
 - switch, 105-111, 121
 - ternary operator, 112
 - empty, in loops, 114
 - expressions, 51-52
 - definition of, 38
 - operators. *See* operators
 - return values, 38, 52
 - finally, 199-202
 - import, 171-172, 175, 256
 - loops
 - breaking, 119
 - do, 118-119
 - for, 113-116
 - index values, 114
 - labeling, 120
 - restarting, 119-120
 - while, 116-118
 - package, 32, 63, 175
 - termination character, 38
- static keyword, 24, 74, 126-127, 134
- static methods, 164-167. *See also* class methods
- static modifier, 158, 164
- static variables, 75, 164-167
- step command (jdb), 654
- stop at command (jdb), 653
- Stop command
 - (appletviewer), 643
- stop in command (jdb), 653
- stopping threads, 217-218
- Storefront application, 181-187
- Storefront.java, 184
- storefronts (online), creating, 181-187

- storing
 - command-line arguments, 111
 - data, 427
- streams, 419-420
 - buffers, 427-431, 484-486
 - byte buffers, 486
 - character sets, 487-488
 - byte streams, 420-422
 - file input streams, 422-425
 - file output streams, 425-427
 - channels, 488-491
 - nonblocking I/O network connections, 492-499
 - character streams, 420, 437
 - reading text files, 437-440
 - writing text files, 440-441
 - console input, 431-432
 - creating, 420
 - data streams, 433-436
 - exception handling, 421-422
 - filtering, 421, 427
 - input, 371
 - opening over Internet, 470-475
 - Path objects, 441-444
 - reading, 420
 - writing to, 421
- string arithmetic, 60-61
- StringChecker.java, 76
- String class
 - selecting substrings, 201
 - valueOf() method, 79
- string data type (XML-RPC), 550
- string literals, 50-51
- String objects
 - concatenation operator, 82
 - creating, 21
- strings
 - in Android apps, 575-577
 - comparing, 88-89
 - concatenating, 60-61
 - converting to numbers, 86
 - dividing into tokens, 69-70
 - formatting, 77-78
 - handling, 82
 - length of, 91
 - switch statements, 121
- strings.xml, 575-579
- StringTokenizer class, 69
- StringTokenizer objects, creating, 69-70
- stringWidth() method, 373
- strokes (drawing), 380-381
- struct data type (XML-RPC), 550
- Structured Query Language. *See* SQL
- structures, data. *See* data structures
- styles (fonts), selecting, 371
- styles.xml, 579
- subclasses, 25
 - casting objects, 84-85
 - defining, 26
 - final classes and, 168
 - method inheritance, 163
 - overriding methods, 148-149
 - protected versus default access, 162
 - variable scope, 132
- Subscriptions.java, 277
- subscripts (arrays), 98-99
- substring() method, 77, 201
- substrings, selecting, 201
- subtraction operator, 52
- superclasses, 25
 - casting objects, 84-85
 - indicating, 126
 - methods in, calling, 150
 - modifying, 27
 - overriding methods, 148-149
 - variable scope, 132
- super keyword, 150
- super() method, 151
- surfaces for drawing, creating, 368-369
- SurveyFrame.java, 333
- SurveyWizard.java, 331
- suspend command (jdb), 656
- Swing, 255, 285, 310
 - applications
 - creating interface, 257-259
 - developing framework, 260-261
 - improving performance, 407-412
 - components, 256, 264
 - adding to containers, 256, 262-263
 - AWT components versus, 256
 - check boxes, 272-274
 - combo boxes, 274-276
 - creating, 256, 261-262
 - dialog boxes, 286-293
 - disabled, 264
 - drop-down lists, 274-276
 - hiding, 264
 - image icons, 265-267
 - labels, 267
 - lists, 276-278
 - menus, 303-307
 - progress bars, 300-303
 - radio buttons, 272-274
 - resizing, 264
 - scrolling panes, 271-272, 296-297
 - sliders, 294-296
 - tabbed panes, 307-310
 - text areas, 269-271
 - text fields, 268
 - toolbars, 297-300
 - containers, panels, 262
 - event-handling, 339. *See also* event listeners
 - action events, 345-346
 - component setup, 341-342

- focus events, 346-349
- item events, 349-351
- keyboard events, 351-352
- methods, 342-345
- mouse events, 352
- mouse movement events, 352-357
- window events, 357
- Info application, 292
- layout managers, 314-315
 - alternatives to, 334
 - border layout, 322-324
 - box layout, 317-319
 - card layout, 325-333
 - combining, 324-325
 - creating, 314
 - flow layout, 315-317
 - grid layout, 320-321
 - insets, 333-334
- SwingConstants interface, 267, 294, 309
- SwingWorker class, 407-413
- switch statements, 105-111, 121
- SWT (Standards Widget Toolkit), 310
- synchronized modifier, 158
- System class, 79, 657
 - class methods, 134
 - in variable (input stream), 431
- System.out class variable, 79
- System.out.format() method, 77-78
- System.out.println() method, 46
- System.out.print() method, 46
- system properties, 656-658

T

- tabbed panes, 307-310
- tables in databases
 - creating, 517
 - viewing, 507-508
- TabPanels.java, 308
- Tag command (appletviewer), 644
- tags
 - javadoc, 647, 650
 - XML, 401, 527-528
- TCP sockets, 475-479
 - client-side
 - closing, 476
 - opening, 475
 - server-side, 479-480
 - designing server applications, 480-482
 - testing server applications, 482-483
- telnet, running, 482
- terminating. *See* stopping
- termination character, 38
- ternary operator, 59, 112
- test variables, switch statements, 106
- testing
 - with loops, 113
 - server applications, 482-483
 - unit testing, 219
- text, drawing, 370-372
 - antialiasing, 372
 - finding font information, 372-375
- text areas, 255, 269-271
- Text class, 533
- text editors
 - associating .java files with, 630
 - selecting, 628-629
- text fields, 255, 268
 - event handling
 - action events, 345-346
 - item events, 349-351
 - password fields, obscuring, 268
- text files
 - reading, 437-440
 - writing, 440-441
- TextFrame.java, 373
- this keyword, 130-131, 145, 327
- Thread class, 191, 211
- threaded applications
 - example, 213-217
 - writing, 211-213
- threads, 191, 211
 - creating, 212
 - running, 213
 - stopping, 217-218
- threads command (jdb), 656
- Throwable class, 194-195
- throwing exceptions, 194, 202, 207
 - checked, 204
 - inheritance issues, 206
 - nested handlers, 208-209
 - throws clause, 203
 - unchecked, 204
- throws keyword, 203-205
- timeout values (sockets), 475
- TimeServer.java, 481
- TimeZone class, 657
- TitleBar.java, 343
- title tag, 401
- T-Mobile G1, 570
- toFile() method, 441
- tokens, 69-70
- TokenTester.java, 69
- Tolksdorf, Robert, 640
- toolbars, 297-300
- tools
 - development, selecting, 616
 - functions, modifying, 638
- toPath() method, 442
- toUpperCase() method, 77
- toXML() method, 534
- Translations editor, 576
- Transmission Control Protocol. *See* TCP sockets
- transport-layer sockets, 480
- trimToSize() method, 235
- troubleshooting. *See also* debugging
 - Android apps, 578, 610
 - checking BIOS settings, 614
 - installing HAXM, 611-613

- arrays, 99
- command-line
 - arguments, 139
- compiled classes, 601
- compiling, 633
- database connections, 514
- errors, fatal, 196
- for loops, 114
- Java Development Kit (JDK) configuration, 624-628
- methods, overloaded, 140
- in NetBeans, 603
- running Android apps, 591
- variables
 - class variables, 75
 - scope, 131
- true value (Boolean), 49
- try and catch blocks, 196-199
 - finally clause, 199-202
- Twitter, author contact information, 608

U

- UIManager class, 259
- UltraEdit, 629
- unboxing, 87
- unchecked exceptions, 194, 204
- underscore (_), in large number literals, 48
- Unicode character set, 40, 49-50, 420, 437, 487
- Unicode Consortium website, 51
- unit testing, 219
- unsigned bytes, 434
- up command (jdb), 655
- updating
 - NetBeans, 598
 - progress bars, 301
- URL (uniform resource locator), 471
- URL class, 471
- URL() constructor, 471
- URL objects, creating, 471

- USB drivers, installing, 591
- user coordinate space, 378
- user interface. *See* GUI
- UTF-8 character set, 487
- UTF-16 character set, 487
- UTF-16BE character set, 487
- UTF-16LE character set, 487
- utilities
 - appletviewer, 642-646
 - command line, 638-639
 - jar, 650-652
 - jarsigner, 658-659
 - javac compiler, 641-642
 - javadoc, 646-650
 - java interpreter, 639-641
 - jdb debugger, 652-653
 - advanced commands, 655-656
 - applet debugging, 655
 - application debugging, 653-655
 - keytool, 658-659

V

- validating XML, 529
- valueOf() method, 79
- values
 - assigning to variables, 43
 - class variables, modifying, 75
 - of instance variables, modifying, 73-74
 - passing arguments by, 132
 - shared values, defining, 43
- variables
 - access control, 159
 - comparison of types, 163
 - default access, 159
 - private access, 159-161
 - protected access, 162
 - public access, 161
 - array variables, 96-97
 - assigning values, 40, 43
 - casting, definition of, 83

- CLASSPATH
 - Windows 7-10, 633-635
 - Windows 98/Me, 635-636
- class variables, 18, 39, 72, 127
 - accessing, 165
 - accessing values, 75
 - defining, 74
 - initial values, 40
 - instance variables versus, 33, 74
 - modifying values, 75
 - troubleshooting, 75
- constant variables, 43-44
- creating, 39-40, 44-45
- declaring, 39-40
- decrementing, 55-56
- definition of, 38
- encapsulation, 159
- environment variables, 656
- final variables, 167
- incrementing, 55-56
- in (input stream), 431
- instance variables, 17, 39, 72
 - accessing values, 72-73
 - class variables versus, 33, 74
 - defining, 21, 126-127
 - initial values, 40
 - length, 99
 - modifying values, 73-74
 - nesting with method calls, 79
- interface type, 178
- in interfaces, 179
- local variables, 39
- naming, 40-41
- overflow, 62
- PATH
 - Windows 7-10, 625-627
 - Windows 98/Me, 627-628
- protecting, 170

- scope, 103, 131-132
 - lexical scope, 121
 - troubleshooting, 131
- static, 164-167
- test variables, switch statements, 106
- types, 41
 - array elements, 98
 - casting, 82-84
 - class types, 43
 - converting to/from objects, 86-87
 - data types, 42-43
 - objects versus, 91
- Variables.java, 45
- Vector class, 235
- vendor tag, 401
- verbose compiler, 642
- version numbers
 - Android, 592
 - Java Development Kit (JDK), 624
- @version tag (javadoc), 647
- viewing
 - database connection information, 510
 - documents (HTML), 643
 - Java documentation, 47
 - tabbed panes, 307
 - tables in databases, 507-508
- visible frames, 258
- void data type, 43
- void return type (methods), 129
- volatile modifier, 158

W

- Weather.java, 53
- web-launched applications (Java Web Start), 392-395
 - configuring web servers for, 405
 - creating JNLP files, 396-404
 - description tag, 406

- icon tag, 406-407
 - security, 405-406
- WebReader.java, 471
- WebServer class, 559
- WebServer() constructor, 559
- web services, 549. *See also* XML-RPC
- well-formed XML, 528
- while loops, 116-118
- white space, adding to XML documents, 540-542
- windowActivated() method, 357
- WindowAdapter class, 358, 456
- windowClosed() method, 357
- windowClosing() method, 357
- windowDeactivated() method, 357
- windowDeiconified() method, 357
- window events, 340, 357
- windowIconified() method, 357
- WindowListener event listener, 340, 357
- WindowListener interface, 455
- windowOpened() method, 357
- windows
 - absolute component placement, 334
 - frames
 - closing, 259-260
 - developing framework, 260-261
 - displaying, 258
 - locations, 258
 - sizing, 257
 - visible, 258
 - layout managers. *See* layout managers
- Windows
 - command-line interface, 619-621
 - creating folders, 622-623
 - opening folders, 621-622
 - running programs, 623

- installing Java Development Kit (JDK), 617-619
- Java programs
 - compiling in, 631-632
 - running in, 631-632
- Windows 7-10
 - CLASSPATH variable, 633-635
 - PATH variable, 625-627
- Windows 98/Me
 - CLASSPATH variable, 635-636
 - PATH variable, 627-628
- wizard interfaces, 327
- Word, 629
- WordPad, 628
- word processors. *See* text editors
- workbench.rss, 526
- wrap() method, 484
- wrapper classes, 134
- writeBoolean() method, 433
- writeByte() method, 433
- writeDouble() method, 433
- writeFloat() method, 433
- writeln() method, 433
- writeLong() method, 433
- write() method
 - buffered character streams, 441
 - buffered output streams, 428
 - character streams, 440
 - char versus int data types, 445
 - file output streams, 425
 - filters, 421
 - streams, 421
 - XML documents, 541
- Writer class, 437
- writeShort() method, 433
- writing
 - applications, threaded, 211-217
 - apps in Android Studio, 575-577

- buffered character streams, 440
- to buffered output streams, 428
- bytes, multiple, 425
- database records, 514-521
- data output streams, 433
- Java code in Android Studio, 584-591
- to streams, 421
- text files, 440-441

X-Z

- XML (Extensible Markup Language), 525
 - advantages, 526
 - dialects, designing, 528-529
 - documents
 - creating, 532-535
 - formatting, 540-542
 - modifying, 536-540
 - files, editing, 576-577
 - processing
 - evaluating XOM, 542-545
 - with Java, 530
 - with XOM, 530-532
 - reason for name, 546
 - RSS and, 526
 - tags, 401, 527-528
 - validating, 529
 - well-formed XML, 528
- XML Object Model. *See* XOM
- XML-RPC, 549-551
 - Apache XML-RPC
 - data types supported, 565
 - installing, 554-556
 - clients, 556-559
 - data types supported, 550
 - debuggers, 554
 - requests
 - responding to, 553-554
 - sending, 551-552
 - servers, 559-564

- XmlRpcClient class, 556
- XmlRpcServer class, 559
- ?xml tag, 527
- XOM (XML Object Model), 530-532
 - adding to NetBeans, 532
 - creating XML documents, 532-535
 - evaluating, 542-545
 - formatting XML documents, 540-542
 - licensing, 531
 - modifying XML documents, 536-540
 - principles, 531
- XOR operator, 58
- XYZ color system, 375
- zero-based methods, 77