

Rogers Cadenhead

*"If you get only one Java book, it should be
Sams Teach Yourself Java In 21 Days."*

—PC Magazine

Sixth Edition

**Covers Java 7
and Android**

Sams **Teach Yourself**

Java™

in **21 Days**



SAMS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Rogers Cadenhead

Sams **Teach Yourself**

Java

in **21 Days**

Sixth Edition

SAMS

800 East 96th Street, Indianapolis, Indiana 46240

Sams Teach Yourself Java in 21 Days, Sixth Edition

Copyright © 2013 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: 978-0-672-33574-7

ISBN-10: 0-672-33574-3

Library of Congress Cataloging-in-Publication Data:

Cadenhead, Rogers.

Sams teach yourself Java in 21 days : covering Java 7 and Android /
Rogers Cadenhead.—6th ed.

p. cm.

ISBN 978-0-672-33574-7 (pbk.)

1. Java (Computer program language) 2. Android (Electronic resource)

I. Title.

QA76.73.J38C315 2013

005.13'3--dc23

2012022262

Printed in the United States of America

Second Printing July 2013

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 or from the use of the programs accompanying it.

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales

1-800-382-3419

corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales

international@pearsoned.com

Editor-in-Chief

Mark Taub

Acquisitions Editor

Mark Taber

Development Editor

Songlin Qiu

Managing Editor

Kristy Hart

Project Editor

Anne Goebel

Copy Editor

Gayle Johnson

Indexer

Tim Wright

Proofreader

Chrissy White,
Language Logistics,
LLC

Technical Editor

Boris Minkin

Editorial Assistant

Vanessa Evans

Cover Designer

Anne Jones

Compositor

Nonie Ratcliff

Contents at a Glance

Introduction	1
WEEK 1: The Java Language	
1 Getting Started with Java	9
2 The ABCs of Programming	37
3 Working with Objects	65
4 Lists, Logic, and Loops	91
5 Creating Classes and Methods	119
6 Packages, Interfaces, and Other Class Features	149
7 Exceptions and Threads	187
WEEK 2: The Java Class Library	
8 Data Structures	221
9 Working with Swing	249
10 Building a Swing Interface	277
11 Arranging Components on a User Interface	305
12 Responding to User Input	339
13 Creating Java2D Graphics	367
14 Developing Swing Applications	391
WEEK 3: Java Programming	
15 Working with Input and Output	419
16 Serializing and Examining Objects	447
17 Communicating Across the Internet	467
18 Accessing Databases with JDBC 4.1 and Derby	499
19 Reading and Writing RSS Feeds	521
20 XML Web Services	545
21 Writing Android Apps for Java	565
Appendixes	
A Using the NetBeans Integrated Development Environment	595
B This Book's Website	603
C Setting Up an Android Development Environment	605
D Using the Java Development Kit	613
E Programming with the Java Development Kit	635

Quiz Answers

Table of Contents

Introduction	1
WEEK 1: The Java Language	7
1 Getting Started with Java	9
The Java Language	10
History of the Language	10
Introduction to Java	11
Selecting a Development Tool	12
Object-Oriented Programming	13
Objects and Classes	14
Attributes and Behavior	17
Attributes of a Class of Objects	17
Behavior of a Class of Objects	18
Creating a Class	19
Running the Program	23
Organizing Classes and Class Behavior	25
Inheritance	25
Creating a Class Hierarchy	27
Inheritance in Action	29
Interfaces	31
Packages	32
Summary	32
Q&A	33
Quiz	34
Questions	34
Certification Practice	34
Exercises	35
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	45
Literals	46
Number Literals	47
Boolean Literals	48
Character Literals	49
String Literals	50
Expressions and Operators	51
Arithmetic	51
More About Assignment	53
Incrementing and Decrementing	54
Comparisons	56
Logical Operators	57
Operator Precedence	58
String Arithmetic	60
Summary	61
Q&A	62
Quiz	62
Questions	62
Certification Practice	63
Exercises	63
3 Working with Objects	65
Creating New Objects	66
Using new	66
How Objects Are Constructed	68
A Note on Memory Management	69
Using Class and Instance Variables	70
Getting Values	70
Setting Values	70
Class Variables	72
Calling Methods	73
Formatting Strings	75
Nesting Method Calls	75
Class Methods	76

References to Objects	77
Casting Objects and Primitive Types	79
Casting Primitive Types	80
Casting Objects	81
Converting Primitive Types to Objects and Vice Versa	83
Comparing Object Values and Classes	84
Comparing Objects	85
Determining the Class of an Object	87
Summary	87
Q&A	88
Quiz	88
Questions	89
Certification Practice	89
Exercises	90
4 Lists, Logic, and Loops	91
Arrays	92
Declaring Array Variables	92
Creating Array Objects	93
Accessing Array Elements	94
Changing Array Elements	95
Multidimensional Arrays	97
Block Statements	98
If Conditionals	99
Switch Conditionals	101
The Ternary Operator	107
For Loops	108
While and Do Loops	111
While Loops	111
Do-While Loops	113
Breaking Out of Loops	114
Labeled Loops	114
Summary	115
Q&A	115
Quiz	116
Questions	116
Certification Practice	117
Exercises	117

5	Creating Classes and Methods	119
	Defining Classes	120
	Creating Instance and Class Variables	120
	Defining Instance Variables	120
	Class Variables	121
	Creating Methods	121
	Defining Methods	122
	The this Keyword	124
	Variable Scope and Method Definitions	125
	Passing Arguments to Methods	126
	Class Methods	127
	Creating Java Applications	129
	Helper Classes	130
	Java Applications and Command-Line Arguments	130
	Passing Arguments to Java Applications	130
	Handling Arguments in Your Java Application	131
	Creating Methods with the Same Name	133
	Constructors	136
	Basic Constructors	137
	Calling Another Constructor	138
	Overloading Constructors	138
	Overriding Methods	140
	Creating Methods That Override Existing Methods	140
	Calling the Original Method	142
	Overriding Constructors	143
	Summary	144
	Q&A	145
	Quiz	146
	Questions	146
	Certification Practice	146
	Exercises	148
6	Packages, Interfaces, and Other Class Features	149
	Modifiers	150
	Access Control for Methods and Variables	151
	Static Variables and Methods	157

Final Classes, Methods, and Variables	159
Variables	159
Methods	160
Classes	160
Abstract Classes and Methods	161
Packages	162
The import Declaration	163
Class Name Conflicts	165
Creating Your Own Packages	165
Picking a Package Name	165
Creating the Folder Structure	166
Adding a Class to a Package	166
Packages and Class Access Control	166
Interfaces	167
The Problem of Single Inheritance	167
Interfaces and Classes	168
Implementing and Using Interfaces	168
Implementing Multiple Interfaces	169
Other Uses of Interfaces	169
Creating and Extending Interfaces	170
New Interfaces	170
Methods Inside Interfaces	171
Extending Interfaces	172
Creating an Online Storefront	172
Inner Classes	179
Summary	181
Q&A	182
Quiz	183
Questions	183
Certification Practice	183
Exercises	185
7 Exceptions and Threads	187
Exceptions	188
Exception Classes	190

Managing Exceptions	191
Exception Consistency Checking	191
Protecting Code and Catching Exceptions	192
The finally Clause	195
Declaring Methods That Might Throw Exceptions	198
The throws Clause	199
Which Exceptions Should You Throw?	200
Passing on Exceptions	201
throws and Inheritance	202
Creating and Throwing Your Own Exceptions	203
Throwing Exceptions	203
Creating Your Own Exceptions	204
Combining throws, try, and throw	204
When and When Not to Use Exceptions	205
When to Use Exceptions	205
When Not to Use Exceptions	206
Bad Style Using Exceptions	206
Threads	207
Writing a Threaded Program	207
A Threaded Application	209
Stopping a Thread	214
Summary	215
Q&A	215
Quiz	217
Questions	217
Certification Practice	217
Exercises	218
WEEK 2: The Java Class Library	219
8 Data Structures	221
Moving Beyond Arrays	222
Java Structures	222
Iterator	224
Bit Sets	225
Array Lists	228

Looping Through Data Structures	231
Stacks	233
Map	235
Hash Maps	236
Generics	241
Summary	245
Q&A	245
Quiz	246
Questions	246
Certification Practice	246
Exercises	247
9 Working with Swing	249
Creating an Application	250
Creating an Interface	251
Developing a Framework	254
Creating a Component	255
Adding Components to a Container	256
Working with Components	258
Image Icons	259
Labels	261
Text Fields	262
Text Areas	263
Scrolling Panes	265
Check Boxes and Radio Buttons	266
Combo Boxes	269
Lists	271
Summary	273
Q&A	273
Quiz	274
Questions	274
Certification Practice	274
Exercises	275
10 Building a Swing Interface	277
Swing Features	278
Standard Dialog Boxes	278
Using Dialog Boxes	283

Sliders	286
Scroll Panes	288
Toolbars	289
Progress Bars	292
Menus	295
Tabbed Panes	299
Summary	301
Q&A	301
Quiz	302
Questions	302
Certification Practice	303
Exercises	304
11 Arranging Components on a User Interface	305
Basic Interface Layout	306
Laying Out an Interface	306
Flow Layout	307
Box Layout	309
Grid Layout	311
Border Layout	314
Mixing Layout Managers	316
Card Layout	317
Using Card Layout in an Application	319
Grid Bag Layout	325
Designing the Grid	327
Creating the Grid	329
Cell Padding and Insets	334
Summary	335
Q&A	335
Quiz	336
Questions	336
Certification Practice	337
Exercises	338
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
Certification Practice	363
Exercises	365
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	375
Testing and Setting the Current Colors	376
Drawing Lines and Polygons	377
User and Device Coordinate Spaces	377
Specifying the Rendering Attributes	378
Creating Objects to Draw	381
Drawing Objects	384
Summary	387
Q&A	387
Quiz	388
Questions	388
Certification Practice	388
Exercises	389

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	406
	Improving Performance with SwingWorker	408
	Summary	413
	Q&A	414
	Quiz	414
	Questions	414
	Certification Practice	415
	Exercises	416
	WEEK 3: Java Programming	417
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	426
	Byte Filters	427
	Character Streams	436
	Reading Text Files	436
	Writing Text Files	439
	Files and Paths	440
	Summary	443
	Q&A	443
	Quiz	444
	Questions	444
	Certification Practice	445
	Exercises	446

16	Serializing and Examining Objects	447
	Object Serialization	448
	Object Output Streams	449
	Object Input Streams	452
	Transient Variables	455
	Checking an Object's Serialized Fields	456
	Inspecting Classes and Methods with Reflection	457
	Inspecting and Creating Classes	457
	Working with Each Part of a Class	459
	Inspecting a Class	461
	Summary	463
	Q&A	463
	Quiz	464
	Questions	464
	Certification Practice	465
	Exercises	465
17	Communicating Across the Internet	467
	Networking in Java	468
	Opening a Stream Over the Net	468
	Sockets	473
	Socket Servers	476
	Testing the Server	479
	The java.nio Package	481
	Buffers	481
	Channels	485
	Summary	495
	Q&A	496
	Quiz	496
	Questions	496
	Certification Practice	497
	Exercises	498
18	Accessing Databases with JDBC 4.1 and Derby	499
	Java Database Connectivity	500
	Database Drivers	501
	Examining a Database	501
	Reading Records from a Database	503

Writing Records to a Database	509
Moving Through Resultsets	516
Summary	517
Q&A	518
Quiz	518
Questions	518
Certification Practice	519
Exercises	519
19 Reading and Writing RSS Feeds	521
Using XML	522
Designing an XML Dialect	525
Processing XML with Java	526
Processing XML with XOM	526
Creating an XML Document	528
Modifying an XML Document	532
Formatting an XML Document	536
Evaluating XOM	538
Summary	540
Q&A	541
Quiz	542
Questions	542
Certification Practice	542
Exercises	543
20 XML Web Services	545
Introduction to XML-RPC	546
Communicating with XML-RPC	547
Sending a Request	548
Responding to a Request	549
Choosing an XML-RPC Implementation	550
Using an XML-RPC Web Service	552
Creating an XML-RPC Web Service	555
Summary	560
Q&A	561
Quiz	561
Questions	561

Certification Practice	562
Exercises	563
21 Writing Android Apps for Java	565
The History of Android	566
Writing an Android App	567
Organizing an Android Project	569
Creating the Program	570
Using an Android Emulator	573
Creating a Debug Configuration	574
Running the App	575
Designing an Android App	577
Preparing Resources	578
Configuring a Manifest File	582
Designing the Graphical User Interface	582
Writing Code	585
Summary	591
Q&A	591
Quiz	592
Questions	592
Certification Practice	592
Exercises	592
Appendixes	593
A Using the NetBeans Integrated Development Environment	595
B This Book's Website	603
C Setting Up an Android Development Environment	605
D Using the Java Development Kit	613
E Programming with the Java Development Kit	635
Quiz Answers	659

About the Author

Rogers Cadenhead is a programmer and author. He has written more than 20 books on programming and web publishing, including *Sams Teach Yourself Java in 24 Hours*. 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 mom, Gail Cadenhead. I'm disappointed you abandoned the beehive hairdo you had in the '60s, but that's the last time you ever disappointed me in the 45 years of my life. Thank you for the room and board, for the love and support, and for introducing me to Ryan's Hope and One Life to Live when I was 8.

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, Gayle Johnson, Songlin Qiu, Anne Goebel, 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 Shulenburg, 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: feedback@sampublishing.com

Mail: Sams Publishing
ATTN: Reader Feedback
800 East 96th Street
Indianapolis, IN 46240
USA

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 *Cupcake Wars* 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 17 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. It’s also found on Android cell phones running popular apps such as Angry Birds and Words with Friends.

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, personal digital assistants, mobile devices, and many other environments. It’s even making a comeback in the browser with sophisticated applications created in Java that are deployed using the Google Web Toolkit.

Now in its eighth major release—Java 7—the Java language has matured into a full-featured competitor to other general-purpose development languages, such as C++, Python, Ruby, and Visual Basic.

You might be familiar with Java programming tools such as Eclipse, Borland JBuilder, and NetBeans. 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 at <http://oracle.com/technetwork/java>, 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 <http://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 the Java Standard Edition, the most widely used version of the language and Java Class Library. 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 and threads, and the ability to run different parts of a program simultaneously.

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, 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. Your programs adopt the Nimbus look and feel introduced in Java 7.
- 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. It also describes `SwingWorker`, a class that improves application performance by using threads.

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 introduces object *serialization*, a way to make objects exist even when no program is running. You learn how to save them to a storage medium such as a hard disk, read them into a program, and then use them again as objects.
- 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 4.1. 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 Eclipse 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: It looks like this.

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. Answers to the questions can be found at the end of the book. 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 primarily an object-oriented programming language. When you do work in Java, you 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 the following topics are covered:

- 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,” classes are templates used to create 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 on Day 2, “The ABCs of Programming,” you learned that using a string literal (a series of characters enclosed in double quotation marks) creates 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 were a primitive data type. This shortcut is unavailable for other classes. To create instances for them, 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");
VolcanoRobot robbie = new VolcanoRobot();
```

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

The following examples show 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*. (You learn more about constructors 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.

Here's an example 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`, and enter the code shown 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: import java.util.StringTokenizer;  
2:  
3: class TokenTester {  
4:  
5:     public static void main(String[] arguments) {  
6:         StringTokenizer st1, st2;  
7:  
8:         String quote1 = "GOOG 604.43 -0.42";  
9:         st1 = new StringTokenizer(quote1);  
10:        System.out.println("Token 1: " + st1.nextToken());  
11:        System.out.println("Token 2: " + st1.nextToken());  
12:        System.out.println("Token 3: " + st1.nextToken());  
13:  
14:        String quote2 = "RHT@60.39@0.78";  
15:        st2 = new StringTokenizer(quote2, "@");  
16:        System.out.println("\nToken 1: " + st2.nextToken());
```

LISTING 3.1 Continued

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

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:

Output ▼

```
Token 1: GOOG
Token 2: 604.43
Token 3: -0.42
```

```
Token 1: RHT
Token 2: 60.39
Token 3: 0.78
```

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 9). 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 10–12 display the values of all three tokens: "GOOG", "604.43", and "-0.42".

The second `StringTokenizer` object in this example has two arguments when it is constructed in line 14—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 15 contains three tokens: "RHT", "60.39", and "0.78".

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 special method is called a constructor.

A *constructor* is a special 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 arguments. When you use *new*, you can specify different arguments in the argument list, and the correct constructor for those arguments is called.

Multiple constructor definitions enable the `TokenTester` 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.

A Note on Memory Management

If you are familiar with other object-oriented programming languages, you might wonder whether the *new* statement 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. Java 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 the object. 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 Java virtual machine periodically looks for unused objects and reclaims the memory that those objects are using. This process is called *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.

Using Class and Instance Variables

At this point, you can create your own object with class and instance variables defined in it, 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 use 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 local 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` local variable.

Setting Values

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

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

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

The `PointSetter` application shown 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 then type the source code shown in Listing 3.2 and save the file.

LISTING 3.2 The Full Text of `PointSetter.java`

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

When you run this application, the output is the following:

Output ▼

```
Starting location:
X equals 4
Y equals 13

Moving to (7, 6)

Ending location:
X equals 7
Y equals 6
```

In this application, you first create an instance of `Point` where x equals 4 and y equals 13 (line 6). These individual values are retrieved using dot notation.

The value of `x` is changed to 7 and `y` to 6. Finally, 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.

For this reason, it's a good idea to use the name of the class when you refer to a class variable. It makes your code easier to read and makes strange results easier to debug.

Calling Methods

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`.

LISTING 3.3 The Full Text of `StringChecker.java`

```
1: class StringChecker {
2:
3:     public static void main(String[] arguments) {
4:         String str = "Would you like an apple pie with that?";
5:         System.out.println("The string is: " + str);
6:         System.out.println("Length of this string: "
7:             + str.length());
8:         System.out.println("The character at position 6: "
9:             + str.charAt(6));
10:        System.out.println("The substring from 26 to 32: "
11:            + str.substring(26, 32));
12:        System.out.println("The index of the first 'a': "
13:            + str.indexOf('a'));
14:        System.out.println("The index of the beginning of the "
15:            + "substring \"apple\": " + str.indexOf("apple"));
16:        System.out.println("The string in uppercase: "
17:            + str.toUpperCase());
18:    }
19: }
```

Save and run the file to display this output:

Output ▼

```
The string is: Would you like an apple pie with that?
Length of this string: 38
The character at position 6: y
The substring from 26 to 32: e with
The index of the first 'a': 15
The index of the beginning of the substring "apple": 18
The string in uppercase: WOULD YOU LIKE AN APPLE PIE WITH THAT?
```

In line 4, you create a new instance of `String` by using a string literal. The remainder of the program simply calls different string methods to do different operations on that string:

- Line 5 prints the value of the string you created in line 4: “Would you like an apple pie with that?”
- Line 7 calls the `length()` method in the new `String` object. This string has 38 characters.
- Line 9 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 `y`.
- Line 11 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 13 calls the `indexOf()` method, which returns the position of the first instance of the given character (here, `'a'`). Character literals are surrounded by single quotation marks; if double quotation marks had surrounded the `'a'` in line 13, the literal would be considered a `String`.
- Line 15 shows a different use of the `indexOf()` method, which takes a string argument and returns the index of the beginning of that string.
- Line 17 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 `y` 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 'W' is at position 0, 'o' at position 1, 'u' at position 2 and so on. This is something you encounter often in Java.

Formatting Strings

Numbers such as money often need to be displayed in a precise manner. There's only two places after the decimal (for the cents), a dollar sign (\$), and commas.

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` return

values that are displayed. The `charAt()` method returns 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`, the uppercase version of `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 several 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 screen 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, like class variables, apply to the class as a whole and not to its instances. 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 different 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 in the discussion of 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 = "item";
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 the difference, the `RefTester` application shown in Listing 3.4 shows how references work. Create an empty Java file for the class `RefTester` in NetBeans, and enter the code shown in Listing 3.4 as the application's source code.

LISTING 3.4 The Full Text of RefTester.java

```
1: import java.awt.Point;
2:
3: class RefTester {
4:     public static void main(String[] arguments) {
5:         Point pt1, pt2;
6:         pt1 = new Point(100, 100);
7:         pt2 = pt1;
8:
9:         pt1.x = 200;
10:        pt1.y = 200;
11:        System.out.println("Point1: " + pt1.x + ", " + pt1.y);
12:        System.out.println("Point2: " + pt2.x + ", " + pt2.y);
13:    }
14: }
```

Save and run the application. Here is the output:

Output ▼

```
Point1: 200, 200
Point2: 200, 200
```

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

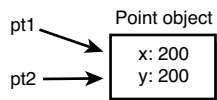
- **Line 5**—Two Point variables are created.
- **Line 6**—A new Point object is assigned to pt1.
- **Line 7**—The value of pt1 is assigned to pt2.

Lines 9–12 are the tricky part. The x and y variables of pt1 both are set to 200, and then all variables of pt1 and pt2 are displayed onscreen.

You might expect pt1 and pt2 to have different values. However, the output shows this not to be the case. As you can see, the x and y variables of pt2 also are 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 3.1. Either variable can be used to refer to the object or to change its variables.

FIGURE 3.1
References to
objects.



If you wanted `pt1` and `pt2` to refer to separate objects, you could use separate `new Point()` statements on lines 6 and 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 Morris, the perpetually hard-to-please cat in the old 9Lives cat food commercials, 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 the same type.

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 data in Java.

Sometimes you'll 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 the concept of casting is reasonably simple, the usage is complicated by the fact that Java has both primitive types (such as `int`, `float`, and `boolean`) and object types (`String`, `Point`, `ZipFile`, 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 primitive type. This most commonly occurs with the numeric types. But one primitive type can never 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`; you can use 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 an `int` or 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 `i` has the value 65, the cast `(char) i` 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` in the following expression:

```
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 easily 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 same 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.

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. Note that 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
```

As you'll see when you begin working with graphical user interfaces during Week 2, "The Java Class Library," 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 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 under any circumstance is cast from an object to a primitive data type, or vice versa.

Primitive types and objects are very 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 that would work:

```
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, and *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.

This feature was unavailable in the first several versions of the language.

Here's an example of autoboxing and unboxing:

```
Float f1 = new Float(12.5F);  
Float f2 = new Float(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 to see whether an object is an instance of a given class

Comparing Objects

Yesterday, you learned about operators for comparing values—equal to, 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, a method of the class called `equals()` 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 and save the file, either by choosing File, Save or by clicking the Save All toolbar button.

LISTING 3.5 The Full Text of `EqualsTester.java`

```
1: class EqualsTester {
2:     public static void main(String[] arguments) {
3:         String str1, str2;
4:         str1 = "Free the bound periodicals.";
5:         str2 = str1;
6:
7:         System.out.println("String1: " + str1);
8:         System.out.println("String2: " + str2);
9:         System.out.println("Same object? " + (str1 == str2));
10:
11:         str2 = new String(str1);
12:
```

LISTING 3.5 Continued

```
13:         System.out.println("String1: " + str1);
14:         System.out.println("String2: " + str2);
15:         System.out.println("Same object? " + (str1 == str2));
16:         System.out.println("Same value? " + str1.equals(str2));
17:     }
18: }
```

Here's the output:

Output ▼

```
String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? true
String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? false
Same value? true
```

The first part of this program declares two variables (`str1` and `str2`), assigns the literal “Free the bound periodicals.” to `str1`, and then assigns that value to `str2` (lines 3–5). As you learned earlier, `str1` and `str2` now point to the same object, and the equality test at line 9 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 (line 15) returns the expected answer: `false`. They are not the same object in memory. Testing them using the `equals()` method in line 16 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 knows enough to give 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 what an object's class is? Here's how you do so 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 objects. It 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
```

```
Point pt = new Point(10, 10);  
boolean check2 = pt 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 defined as some form of punctuation. 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 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.

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 usually 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 38 characters long, shouldn't the characters be numbered from 1 to 38 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 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. Answers are at the end of the book.

Questions

1. Which 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.

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 04/29/2013) 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.

Where applicable, exercise solutions are offered on the book's website at www.java21days.com.

Index

Symbols

- [[]], 98
- {}, 38
- & (ampersand), AND operators, 57
- [] (brackets), 92
- ^ (caret), XOR operator, 57
- !! command (jdb), 653
- // comment notation, 46
- <> (diamond operator), 244
- = (equal sign)
 - assignment operator, 40, 43
 - equality operator (==), 56, 85
- ! (exclamation point), NOT operator, 58
- > (greater than), 56
- > = (greater than or equal to symbol), 56
- < (less than), 56
- < = (less than or equal to symbol), 56
- (hyphen), decrement operator (—), 55-56
- != (inequality operator), 56, 85
- (minus signs), 47
- O option (javac), 640
- . (period), dot notation, 70
- | (pipe character), OR operators, 57
- + (plus sign)
 - concatenation operator (+), 60
 - increment operator (++), 55-56
- ; (semicolon), statement termination character, 38
- 2D graphics
 - arcs, drawing, 382-383
 - coordinate spaces, 377-378
 - coordinate system, 369
 - ellipses, drawing, 381
 - lines, drawing, 381
 - Map applet, 384-387
 - polygons, drawing, 383-384
 - rectangles, drawing, 381
 - rendering attributes, 378
 - drawing strokes, 380
 - fill patterns, 378-379

A

- abstract classes, 161-162
- abstract methods, 161-162
 - final, 182
 - private, 182
- abstract modifier, 161
- access control, 151, 155
 - accessor methods, 156
 - default access, 151
 - inheritance, 155-156

- packages, 166
 - default access, 166
 - public access, 166-167
 - private access, 152-153
 - protected access, 154-155
 - public access, 153-154
- accessing**
 - array elements, 94-95
 - array list elements, 229
 - class variable values, 72
 - class variables, 72
 - databases, JDBC, 501
 - elements, Map interfaces, 235
 - instance variables, 70
 - methods, class, 157
 - MS-DOS, 618
 - Notepad, 622
 - variables, class, 157
- accessor methods, 156**
 - performance issues, 182
- action events, event handling, 345-346**
- ActionListener event listener, 340**
- actionPerformed() method, 342**
- acyclic gradients, 378**
- adapter classes, 357-359**
- add() method, 229, 256**
- addActionListener() method, 345**
- adding**
 - classes to packages, 166
 - components
 - to containers, 250, 256
 - to panels, 317
 - to toolbars, 290
 - elements, array lists, 229
 - stack elements, 234
- addItem() method, 269**
- AdjustmentListener event listener, 340**
- afterLast() method, 517**
- aligning**
 - components, 306-307
 - border layouts, 314-315
 - card layouts, 317-318
 - flow layouts, 307-309
 - grid bag layouts, 325-327
 - grid layouts, 312
 - panels, 317
 - labels, 261
- AllCapsDemo.java application, 441-442**
- allocate() method, 482**
- allocating memory to objects, 69**
- ampersand (&), AND operators, 57**
- AND operators, 57**
- Android**
 - apps
 - code, writing, 584-589
 - designing, 577
 - GUI, designing, 582-584
 - manifest files, configuring, 581-582
 - resources, preparing, 578-580
 - running, 575-576
 - testing, 611
 - writing, 567-572
 - AVD, 573-574
 - debug configuration, creating, 574-575
 - history of, 566-567
 - project, organizing, 569-570
- Android plug-in for Eclipse, installing, 608-611**
- Android SKD, installing, 607**
- animation, controlling with threads, 207**
- Ant, 182**
- antialiasing, 372**
- AppInfo application, 642**
 - AppInfo.html, 643-644
 - AppInfo.java, 642-643
- AppInfo2 application, 645-647**
- Applet menu commands**
 - Clone, 642
 - Info, 642
 - Reload, 641
 - Restart, 641
 - Start, 641
 - Stop, 641
 - Tag, 642
- applet tag, attributes (archive), 649**
- applets**
 - background color, setting, 377
 - debugging, 653
 - Map, 384-387
- appletviewer browser, 640**
 - AppInfo sample application, 642
 - AppInfo.html, 643-644
 - AppInfo.java, 642-643
 - Applet menu commands, 641-642
- appletviewing browser, 643**
- applications, 129**
 - AllCapsDemo, 441-442
 - Android
 - code, writing, 584-589
 - designing, 577
 - GUI, designing, 582-584
 - manifest files, configuring, 581-582
 - resources, preparing, 578-580
 - running, 575-576
 - testing, 611
 - writing, 567-572
 - AppInfo, 642-644
 - AppInfo2, 645-647
 - arguments
 - handling, 131-132
 - passing to, 130
 - BufferDemo.java, 428-429
 - Buttons.java, 256-257

- ChangeTitle, 343, 345
 - ChangeTitle.java, 343-344
 - ChooseTeam.java, 268-270
 - CopyArrayWhile.java, 112
 - main() method, 112-113
 - output, 112
 - DayCounter.java, 103-104
 - debugging, 651-653
 - DigitalClock.java, 209
 - Finger.java, 475-476
 - Form.java, 263-264
 - HalfDollars
 - main() method, 97
 - source code, 95
 - HalfLoop.java, 110
 - helper classes, 130
 - HexRead.java, 196
 - Icons.java, 259-260
 - Info (dialog box), 283-284
 - ItemProp, 656
 - main() method,
 - signature, 129
 - multitasking, 207
 - RangeClass.java, 123
 - ReadBytes.java, 424
 - ReadPrimes.java, 435
 - ReadSource.java, 438
 - running, 637
 - SelectItem.java, 349-350
 - SimpleFrame.java, 254
 - Slider.java, 287
 - Storefront, 172
 - SumAverage.java, 132
 - Swing, graphical, 251
 - SwingColorTest, 357
 - threaded
 - clock, 209
 - writing, 207-209
 - TimeServer client/server
 - designing, 477-479
 - testing, 479-480
 - ToolBar.java, 290
 - WordPad, 626
 - WriteBytes.java, 425-426
 - applications (Java), 130**
 - creating, 129-130
 - Arc2D.Float class, 382-383**
 - archive attribute (applet tag), 649**
 - archiving files, jar files, 648-650**
 - arcs, drawing (Arc2D.Float class), 382-383**
 - ArgStream() method, 430**
 - arguments**
 - in applications, handling, 131-132
 - command line, 636
 - grouping, 131
 - objects, creating, 66
 - quotation marks in, 131
 - to applications, passing, 130
 - to methods, passing, 126-127
 - arguments (commands), 636**
 - arithmetic, string, 60**
 - arithmetic operators, 51-52**
 - example, 52-53
 - Array class, 459**
 - array elements, grids, 97**
 - array lists, 223, 228-231**
 - capacity, 228
 - creating, 228-229
 - elements, 229-230
 - looping through, 231-233
 - size, 228
 - sizing, 231
 - ArrayIndexOutOfBoundsException exception, 190**
 - ArrayList class, 223, 228-231**
 - arrays, 92, 222**
 - boundaries, 94
 - creating, 95
 - elements
 - accessing, 94-95
 - changing, 95-97
 - data types, 94
 - HalfDollars.java
 - application, 96
 - implementing, 92
 - limitations, 222
 - modifying, 95
 - multidimensional, 97
 - objects, creating, 93-94
 - primitive types, 458
 - references, 95
 - sample program
 - main() method, 97
 - source code, 95
 - subscripts, 94
 - troubleshooting, 95
 - variables, declaring, 92-93
 - ASCII text, 627**
 - assignment variable values, 40, 43**
 - overflow, 62
 - assignment operators, 53-54**
 - equal sign (=), 40
 - associating**
 - components with event listeners, 341-342
 - filters, 421
 - attributes**
 - applet tag, archive, 649
 - classes, 17-18
 - defining, 18
 - objects, defining, 18
 - @author tag (javadoc), 645**
 - AUTOEXEC.BAT, editing, 622**
 - AVD (Android Virtual Device), 573-574**
 - avoiding exceptions, 206**
- ## B
- background color, setting, 377**
 - base-16 numbering system, 47**
 - base-8 numbering system, 47**
 - beforeFirst() method, 517**

behavior

- classes, 18-19
 - organizing, 25-32
 - shared, 31-32
- bits, 223-228**
- BitSet class, 223-228**
- block statements, 38**
- blocks, 98-99**
 - try and catch, 192
 - try...catch, example, 193-194
- boolean data type, 42**
- Boolean literals, 48-49**
- Boolean values, 42**
- border layout manager, 315**
 - creating, 314-315
- BorderLayout() method, 314**
- boundaries, arrays, 94**
- braces (braces), 98**
- brackets ([]), 92**
- break keyword, 114**
- breaking loops, 114**
- breakpoints, 651**
 - deleting, 652-653
 - setting, 652
- browser (appletviewer), 640, 643**
 - AppInfo sample application, 642
 - AppInfo.html, 643-644
 - AppInfo.java, 642-643
 - Applet menu commands, 641-642
- BufferConverter sample application, 486-488**
- BufferDemo.java application, 428-429**
- buffered streams, 427**
 - creating, 427-428
 - reading, 427
 - writing to, 428-430
- BufferedInputStream() method, 427**
- BufferedOutputStream() method, 428**

- BufferedReader objects, 468**
- BufferedReader() method, 437**
- BufferedWriter() method, 440**
- buffers, 427, 481-483**
 - byte buffers, 483-484
 - transforming to/from character buffers, 485
- buffers, 485. See also channels**
- buildRect() method, 134**
- built-in fonts, 371**
- ButtonGroup object, 267**
- buttons, event handling**
 - action events, 345-346
 - item events, 349-350
- Buttons.java, 256-257**
- byte buffers, 483-484**
 - transforming to/from character buffers, 485
- byte data type, 42**
- byte filters, 427**
- byte streams, 420-422, 448. See also streams**
 - file input streams, 422-423
 - file output streams, 425-426
- bytecode, running, 637**
- bytes**
 - multiple, writing, 425
 - reading from channels, 486

C**calling**

- constructor methods, 136
- constructors from another constructor, 138
- methods, 19, 73-74
 - class methods, 77
 - constructors, 143
 - of objects, 76
 - start(), 641
 - stop(), 641
 - in superclasses, 143

- calls, method (nesting), 75-76**
- capacity, array lists, 228**
- card layout manager, 317**
 - creating, 318
- CardLayout() method, 318**
- caret (^), XOR operator, 57**
- case-sensitivity, 41**
- casting. See also converting**
 - boolean values, 80
 - datatypes, 80-81
 - definition of, 80
 - destinations, 80
 - objects, 79-80
 - to classes, 81-82
 - to interfaces, 82, 170
 - primitive types, 79-81
 - sources, 80
 - superclasses, 82
 - variables, 80-81
- casts, explicit, 81**
- catch blocks, Exception classes in, 194**
- catch clauses, empty, 206**
- catch command (jdb), 654**
- catching exceptions, 190-192**
 - finally statement, 195-196
 - try and catch blocks, 192-194
- cd command, 628**
- CD command (MS-DOS), 619**
- ChangeTitle application, 343-345**
- ChangeTitle.java application, 343-344**
- changing array elements, 95-97**
- channel() method, 491**
- channels, 485-488**
 - BufferConverter sample application, 486-488
 - nonblocking I/O, 488-495
 - reading bytes, 486
- char data type, 42**

- character buffers, transforming to/from byte buffers, 485
- character encodings, list of, 484
- character literals, 49. *See also* Unicode character set
 - escape codes, 49-50
- character sets, 484-485
 - creating, 484
 - Unicode, 41
 - escape codes, 49
- character streams, 436, 448. *See also* streams
 - reading text files, 436-439
 - writing text files, 439-440
- charAt() method, 74
- Charset class, 484
- CharsetDecoder class, 484
- CharsetEncoder class, 484
- charWidth() method, 373
- check boxes, 266-267
 - event handling
 - action events, 345-346
 - item events, 349-350
 - nonexclusive, 267
- checkTemperature() method, 25
- ChooseTeam.java, 268-270
- choosing
 - port numbers, 477
 - text editors, 625-626
 - WordPad, 626
 - XML-RPC implementation, 550-551
 - XML-RPC web service, 552-555
- Class class, 457-459
- .class extensions, 637
- class files, specifying, 638
- class keyword, 120
- class methods, 19, 76-77, 127
 - accessing, 157
 - availability, 128
 - calling, 77
 - defining, 128
- Class not found error, 630
- class types, 43
- class variables, 18, 39, 70, 121
 - accessing, 157
 - defining, 40, 72
 - troubleshooting, 73
 - values
 - accessing, 72
 - initial values, 40
 - mod, 72
 - modifying, 72
 - versus instance variables, 72
- classes, 119. *See also* packages
 - abstract, 161-162
 - adding to packages, 166
 - Arc2D.Float, 382-383
 - Array, 459
 - ArrayList, 223, 228-231
 - as subclasses, defining, 26
 - attributes, 17-18
 - behavior, 18-19
 - organizing, 25-32
 - BitSet, 223-228
 - BufferConverter, 486-488
 - Charset class, 484
 - CharsetDecoder class, 484
 - CharsetEncoder class, 484
 - Class, 457-459
 - CodeKeeper, 232
 - CodeKeeper2, 243-244
 - Color, 375
 - ColorSpace, 375
 - ComicBooks, 239-240
 - constants, 43
 - declaring, 44
 - Constructor, 459
 - creating, 19-22, 457-459, 598-599
 - instance variables, 21
 - defining, 15, 120
 - Dictionary, 223
 - Dimension, 252
 - efficiency, increasing, 160
 - Ellipse2D.Float, 381
 - Error, 190
 - exception, 190-193
 - in catch blocks, 194
 - constructors, 204
 - Field, 459
 - FileInputStream, 485
 - FileOutputStream, 485
 - FilterInputStream, 426
 - FilterOutputStream, 426
 - final, 159
 - subclassing, 160
 - final classes, 160-161
 - FingerServer, 492-495
 - FlowLayout, 306-307
 - FontMetrics, 373-375
 - functionality, 25
 - Giftshop, 177-178
 - Graphics2D, 368-369
 - grouping, 32
 - HashMap, 223, 236-241
 - helper classes, 130
 - hierarchies, 26
 - creating, 27-29
 - designing, 28-29
 - HolidaySked, 226-227
 - URLConnection, 472
 - identifying, 162
 - importing, 163
 - InetAddress, 489
 - inheritance, 29-30
 - multiple, 167
 - overview, 27
 - single, 31, 167-168
 - inner, 180
 - inner classes, 179, 388
 - advantages, 181
 - naming, 181
 - scope, 181
 - InputStream, 422

- inspecting, 457-463
- InstanceCounter, 158-159
- instances of, creating, 16
- versus interfaces, 168
- IOException, 191
- Item, 173-174
- Java 2 Class Library, 16
- javax.swing.JButton, 17
- javax.swing.JComponent, 258
- JCheckBox, 266
- JComboBox, 269
- JOptionPane, 278
 - confirm dialog boxes, 279
 - input dialog boxes, 280-281
 - message dialog boxes, 281
 - option dialog boxes, 282-283
- JPanel, 368
- JProgressBar, 293
- JRadioButton, 266
- JScrollPane, 288-289
- JSlider, 286
 - constructors, 286
 - methods, 286-287
 - Slider.java sample application, 287
- JToolBar, 289-290
- Line2D.Float, 381
- Map, 384-386
- Method, 459-460
- MethodInspector, 461
- methods, 19
- Modifier, 459
- modifying, 208
- MyRect, 133
 - buildRect() method, 134
 - definition, 135
- MyRect2, 139
- name conflicts, 165
- NamedPoint, 144
- Object, 26
 - object-oriented programming, 15
 - inheritance, 25-26
 - ObjectInputStream, 452-455
 - ObjectOutputStream, 449-451
 - ObjectReader, 454-455
 - objects to, casting, 81-82
 - ObjectWriter, 451-452
 - of objects, determining, 87
 - organization, 150
 - organizing, 25-32, 162
 - packages, 32
 - adding, 166
 - creating, 638
 - PassByReference, 126
 - PrintClass, 141
 - protecting, 162
 - Random(), enabling, 69
 - Reader, 463
 - Rectangle2D.Double, 381
 - Rectangle2D.Float, 381
 - reflection, 457, 461-463
 - reflection classes, 459-461
 - ScrollPaneConstants, 265
 - SelectionKey, 490-491
 - ServerSocket, 476-477
 - Socket, 473
 - SocketImpl, 477
 - SquareTool, 180
 - Stack, 223, 233-235
 - Storefront, 176-177
 - StringTokenizer, 67
 - subclasses, 25
 - subclassing, 27
 - superclasses, 25
 - indicating, 120
 - modifying, 27
 - SwingConstants, 261, 286
 - TextFrame, 373-374
 - Thread, 207
 - Throwable, 190
 - TimeServer, 478-479
 - top-level, 179
 - defining, 181
 - variables
 - class variables, 121
 - instance variables, 120-121
 - Vector, 231
 - WebReader, 469-470
 - wrapper classes, 128
 - Writer, 463
- classes (object-oriented programming), 14**
- classes command (jdb), 654**
- CLASSPATH variable (MS-DOS)**
 - Windows 2000/XP, 632-633
 - Windows 9x/Me, 630-631
- clauses, catch (empty), 206**
- clear command (jdb), 652**
- clear() method, 230, 237**
- client-side sockets**
 - closing, 474
 - instantiating, 473
 - opening, 473
 - reading/writing, 473
- clock, threaded application, 209**
- Clone command (Applet menu), 642**
- close() method, 421**
- closePath() method, 384**
- closing**
 - ODBC data source connections, 507
 - socket connections, 474
- CoalTotals.java, 508**
- code, writing for Android apps, 584-589**
- code (Java), complexity, 206**
- code listings**
 - AppInfo application
 - AppInfo.html, 643-644
 - AppInfo.java, 642-643

- AppInfo2 application, 645-647
- ItemProp application, 656
- CodeKeeper class, 232**
- CodeKeeper2 class, 243-244**
- collection, garbage, 69**
- color**
 - background colors, 377
 - dithering, 375
 - drawing colors, setting, 376-377
 - finding current color, 377
 - sRGB color system, 375
 - XYZ color system, 375
- Color class, 375**
- Color objects, 375**
- color spaces, 375**
- colors, Color objects, 375**
- ColorSpace class, 375**
- combining**
 - layout managers, 316-317
 - methods, nested, 76
- combo boxes, 269-270**
 - event handling
 - action events, 345-346
 - item events, 349-350
- ComicBooks class, 239-240**
- command line, 636**
 - arguments, 636
 - options, 636-637
- command-line interfaces, 617-618**
- command-line tools, javac, 628**
- commands**
 - Applet menu, 641-642
 - arguments, 636
 - jar, 648-650
 - jdb (debugger)
 - !!, 653
 - catch, 654
 - classes, 654
 - clear, 652
 - cont, 653
 - down, 654
 - exit, 653
 - ignore, 654
 - list, 653
 - locals, 653
 - memory, 654
 - methods, 654
 - print, 653
 - resume, 654
 - run, 652
 - step, 653
 - stop at, 652
 - stop in, 652
 - suspend, 654
 - threads, 654
 - up, 654
 - JDK, format, 636
 - MS-DOS
 - CD, 619, 628
 - CLASSPATH variable, 630-633
 - MD, 620
 - PATH variable, 622-625
 - SET CLASSPATH=, 631
 - Start menu
 - Find, 623
 - Run, 615
- comment notation, 46**
- comments, 45**
 - notation, 46
 - source code, 644
- comparing**
 - instances, 85
 - objects, 85-86
- comparison operators, 56-57**
- compilation errors, 95**
- compiler (javac), 639-640**
- compiler errors, 242**
- compilers, 614**
- compiling**
 - files, 639-640
 - multiple, 639
 - Java programs in Windows, 628-630
 - programs, 617
 - troubleshooting, 630
- complexity, code (Java), 206**
- complications, multiple interfaces, 169**
- components, 258**
 - aligning, 306-307
 - border layouts, 314-315
 - card layouts, 317-318
 - flow layouts, 307-309
 - grid bag layouts, 325-327
 - grid layouts, 312
 - panels, 317
 - associating with event listeners, 341-342
 - check boxes, 266-267
 - combo boxes, 269-270
 - to containers, adding, 250, 256
 - creating, 255-256
 - dialog boxes, 278
 - confirm dialog boxes, 279
 - Info sample application, 283-284
 - input dialog boxes, 280-281
 - message dialog boxes, 281
 - option dialog boxes, 282-283
 - disabled, 258
 - drop-down lists, 269-270
 - icons, 259-261
 - labels, 261-262
 - methods, stop(), 214
 - to panels, adding, 317
 - progress bars, 293
 - radio buttons, 266-267
 - resizing, 258
 - scroll panes, 288-289
 - scrollbars, configuring, 265
 - scrolling panes, 265

- SDK 1.4, 616
- sliders, 286
 - advantages, 286
 - labels, 286-287
 - orientation, 286
 - Slider.java sample application, 287-288
- Swing, creating, 250
- text areas, 263-265
- text fields, 262-263
- toolbars, 289
 - adding, 290
 - dockable toolbars, 290
 - orientation, 289
- windows, frames, 251
- concatenating strings, 60**
- concatenation operator (+), 60**
- Conder, Shane, 589, 607**
- conditional operator, 107-108**
- conditionals, 101, 105-107**
 - conditional operator, 107-108
 - if, 99
 - else keyword, 99
 - examples, 100
 - switch, 102-103
 - limitations, 102
- configureBlocking() method, 490**
- configuring**
 - manifest files for Android apps, 581-582
 - scrollbars, 265
 - SDK 1.4, 617
 - CLASSPATH variable, 630-633
 - command-line interface, 617-618
 - configuration errors, 622
 - MS-DOS folders, 618-619
 - PATH variable, 622-625
 - troubleshooting, 616
- confirm dialog boxes, 279**
- ConfirmDialog dialog boxes, 278**
- conflicts, names (reducing), 162**
- connect() method, 490**
- connections**
 - networking. *See* networking
 - telnet, 479-480
- consistency checking (exceptions), 191-192**
- constant variables, 159**
- constants, 43**
 - declaring, 44
- Constructor class, 459**
- constructor methods, 136-138**
 - calling, 136, 143
 - from another constructor, 138
 - naming, 137
 - overloading, 138-139
 - overriding, 143-144
- constructors**
 - definition of, 69
 - Dimension(), 252
 - exception classes, 204
 - JCheckBox(), 267
 - JFrame(), 251
 - methods
 - setSelected(), 266
 - SimpleFrame(), 255
 - URL(), 469
- constructors (methods), 67**
- cont command (jdb), 653**
- containers**
 - components, adding, 250, 256
 - panels, 256
- contains() method, 230**
- containsKey() method, 237**
- containsValue() method, 237**
- contents, labels, 262**
- continue keyword, 114**
- controlling animation with threads, 207**
- controlling access. *See* access control**
- conventions, naming, 165**
- converting, 80. *See also* casting**
 - objects, 79-80
 - primitive types, 79-80
 - primitive types and objects, 83
 - source code, 639
 - text to uppercase, AllCapsDemo application, 441
- coordinate spaces (graphics), 377-378**
- coordinate system (graphics), 369**
- CopyArrayWhile.java, main() method, 112-113**
- CopyArrayWhile.java application, 112**
- createFont() method, 371**
- createStatement() method, 505**
- creating**
 - array lists, 228-229
 - arrays, 95
 - objects, 93-94
 - character sets, 484
 - classes, 1-22, 457-459
 - packages, 638
 - components, 255-256
 - Swing, 250
 - debug configuration, Android, 574-575
 - drawing surfaces, 368-369
 - exceptions, 204
 - folders in MS-DOS, 620
 - instances of classes, 16
 - interfaces, 170-171, 251-253
 - Java applications, 129-130
 - Java class with NetBeans, 598-599
 - JNLP files, 396-405
 - labels, 261

- layout managers, grid bag
 - layout, 326
- layouts, grid bag, 325
- methods, overloaded, 133-134
- NetBeans project, 596
- objects, 66-68
 - arguments, 66
 - Font, 370
 - ImageIcon, 259
 - with new operator, 66-69
 - String, 21
 - StringTokenizer, 67-68
 - URL, 469
- output streams, 450
- source files, 627-629
- storefronts, online, 172-179
- streams, 420
- threads, 208-209
- variables, 39
- XML-RPC web service, 555-560
- curly braces (curly braces), 38**
- current color, finding, 377**
- current objects, referring to, 124**
- custom packages**
 - access control, 166
 - default access, 166
 - public access, 166-167
 - classes, adding, 166
 - folder structure, 166
 - naming, 165-166
- cyclic gradients, 378**

- D**
- Darcey, Lauren, 589, 607**
- data, storing, 427**
- data sources (ODBC), connections**
 - closing, 507
 - opening, 503-507
- data streams, 432**
 - creating, 432
 - reading, 433
 - ReadPrimes, 435
 - sample applications, WritePrimes, 434
 - WritePrimes, 434
- data structures, 222**
 - array lists, 223, 228-231
 - accessing, 229
 - adding elements, 229
 - capacity, 228
 - changing elements, 230
 - creating, 228-229
 - looping through, 231-233
 - removing elements, 230
 - searching elements, 230
 - size, 228
 - sizing, 231
 - arrays, 222
 - limitations, 222
 - bits, 223-228
 - dictionaries, 223
 - Enumeration interface, 223
 - generics, 241-244
 - hash maps, 223, 236-241
 - clearing, 237
 - creating, 237
 - hash codes, 238
 - load factor, 237
 - Iterator interface, 223-225
 - Java, 222
 - key-mapped, 235-236
 - stacks, 223, 233-235
 - adding elements, 234
 - logical organization, 233
 - popping off elements, 234
 - searching, 235
 - vectors, 231
- data types, 42**
 - boolean, 42
 - char, 42
 - double, 42
 - float, 42
 - integers, 42
 - primitive, 42
 - primitive data types, buffers, 481
- databases**
 - JDBC (Java Database Connectivity), 500-501
 - accessing, 501
 - drivers, 501
 - ODBC data source connections, 503-507
 - queries, 500
 - records, navigating, 507, 517
 - DataInputStream() method, 432**
 - DataOutputStream() method, 433**
 - datatypes, 88**
 - casting, 80-81
 - DayCounter.java application, 103-104**
 - deallocating memory, objects, 69**
 - debug configuration, Android, 574-575**
 - debugger (jdb), 650-654**
 - applet debugging, 653
 - application debugging, 651-653
 - breakpoints, 651
 - deleting, 652-653
 - setting, 652
 - commands
 - !!, 653
 - catch, 654
 - classes, 654
 - clear, 652
 - cont, 653
 - down, 654
 - exit, 653
 - ignore, 654
 - list, 653

- locals, 653
- memory, 654
- methods, 654
- print, 653
- resume, 654
- run, 652
- step, 653
- stop at, 652
- stop in, 652
- suspend, 654
- threads, 654
- up, 654
- single-step execution, 651
- debugging, 654. See also debugger (jdb)**
 - applets, 653
 - applications, 651-653
 - breakpoints, 651
 - deleting, 652-653
 - setting, 652
 - programs, 626, 650
 - single-step execution, 651
 - Spartacus application, 601
- declarations**
 - import, 163-164
 - package, 152, 166
- declaring**
 - array variables, 92-93
 - arrays of arrays, 97
 - constants, 44
 - interfaces, 168-171
 - methods, buildRect(), 134
 - variables, 39
- decode() method, 485**
- decrement operations, 54**
- decrement operator (—), 55-56**
- decrementing variables, 54**
- default access, 151**
- defining**
 - attributes, 18
 - class variables, 40
 - classes, 15, 120
 - as subclasses, 26
 - top-level, 181
 - hierarchies, 161
 - methods, 122
 - class methods, 128
 - parameter lists, 122
 - this keyword, 124
 - values, shared, 43
 - variables
 - class variables, 72
 - instance variables, 120-121
- definitions**
 - methods, multiple, 133
 - MyRect class, 135
- delete() method, 441**
- deleting**
 - array list elements, 230
 - breakpoints, 652-653
- @deprecated tag (javadoc), 648**
- deprecation option (javac), 640**
- deriveFont() method, 372**
- descriptions, JNLP files, 406**
- designing**
 - apps, Android, 577
 - grids, 327
 - GUI for Android apps, 582-584
 - hierarchies, classes, 28-29
 - socket client/server application, 477-479
 - XML dialects, 525-526
- destinations (casting), 80**
- determining classes of objects, 87**
- developing**
 - frameworks, 254-255
 - Java programs, 614
- development tools, selecting, 12, 614**
- device coordinate space, 378**
- dialects, designing XML, 525-526**
- dialog boxes, 278**
 - confirm dialog boxes, 279
 - ConfirmDialog, 278
 - Info sample application, 283-284
 - input dialog boxes, 280-281
 - InputDialog, 278
 - message dialog boxes, 281
 - MessageDialog, 278
 - option dialog boxes, 282-283
 - OptionDialog, 278
- diamond operator (<>), 244**
- Dictionary class, 223**
- DigitalClock.java application, 209**
- Dimension class, 252**
- Dimension() constructor, 252**
- disabled components, 258**
- displaying**
 - errors, 629
 - frames, 252
- dithering, 375**
- division operators, 52**
- do...while loops, 113**
- dockable toolbars, 290**
- documentation tool (javadoc), 644-648**
 - AppInfo2 sample application, 645-647
 - tags, 645-648
- documentation tools, javadoc**
 - AppInfo2 sample application, 645-647
 - tags, 645, 648
- documents, viewing HTML, 641**
- DomainEditor application, creating, 532, 535**
- DomainWriter application, creating, 536-538**
- DOS text, 627**
- dot notation, evaluating, 70**
- double data type, 42**

- down command (jdb), 654**
- downloading**
 - Eclipse, 606
 - JDK, 636
- drawing**
 - arcs, Arc2D.Float class, 382-383
 - coordinate spaces, 377-378
 - coordinate system, 369
 - ellipses, Ellipse2D.Float class, 381
 - lines, Line2D.Float class, 381
 - polygons, Java2D, 383-384
 - rectangles, Rectangle2D.Float class, 381
 - rendering attributes, 378
 - drawing strokes, 380
 - fill patterns, 378-379
 - text, 370-372
 - antialiasing, 372
 - finding font metrics, 372-375
- drawing colors, setting, 376-377**
- drawing surfaces, creating, 368-369**
- drawString() method, 370**
- drivers, JDBC (Java Database Connectivity), 501**
- drop-down lists, 269-270**
- DTD (Document Type Definition), 525**
- duplicating variables, 126**
- E**
- ea option (javac), 639**
- Eclipse**
 - Android plug-in, installing, 608-611
 - installing, 606
- editing**
 - array list elements, 230
 - AUTOEXEC.BAT, 622
 - system properties, 655
- editors (text), 625-626**
- efficiency, classes (increasing), 160**
- elements**
 - arrays
 - accessing, 94-95
 - changing, 95-97
 - data types, 94
 - grids, 97
 - Map interfaces, accessing, 235
 - XML, 523
- elements (array lists)**
 - accessing, 229
 - adding, 229
 - changing, 230
 - removing, 230
 - searching, 230
- elements (stack)**
 - adding, 234
 - popping off, 234
 - searching, 235
- Ellipse2D.Float class, 381**
- Ellipses, drawing (Ellipse2D.Float class), 381**
- else keyword, 99**
- empty catch clauses, 206**
- empty statements, for loops, 109**
- empty() method, 235**
- emulators, AVD, 573-574**
- enabling**
 - classes, Random(), 69
 - telnet, 480
- encapsulating objects, 153**
- encapsulation, 151**
- encode() method, 485**
- end-of-file exception (EOFException), 433**
- end-of-line characters, 438-440**
- endcap styles (drawing strokes), 380**
- Enumeration interface, 223**
- environment variables, 655**
- EOFException (end-of-file exception), 433**
- EOFException exception, 191**
- equal sign (=)**
 - assignment operator, 40, 43
 - equality operator (==), 56, 85
- equality operator (==), 56, 85**
- equals() method, 85, 238**
- Error class, 190**
- error-handling**
 - methods, 198
 - traditional method, 188-189
- errors, 215. See also exceptions**
 - Class not found, 630
 - compilation, 95
 - compiler, 206-207, 242
 - displaying, 629
 - Error class, 190
 - error-handling, traditional method, 188-189
 - Exception class, 190
 - fatal, troubleshooting, 192
 - NoClassDef, 630
 - runtime, 242
- escape codes, 49-50**
 - Unicode character set, 49
- evaluating**
 - dot notation, 70
 - XOM, 538-540
- event handling**
 - ChangeTitle.java application, 343-344
 - item events, 349-350
- event listeners, 340**
 - ActionListener, 340
 - AdjustmentListener, 340
 - associating components with, 341-342
 - FocusListener, 340
 - methods, 346
 - importing, 341

- ItemListener, 340, 349
 - itemStateChanged() method, 349
 - KeyListener, 340, 351
 - MouseListener, 340, 351
 - MouseMotionListener, 340
 - WindowListener, 340, 357
 - event-handling, 339**
 - action events, 345-346
 - ChangeTitle.java, 343-345
 - event listeners
 - ActionListener, 340
 - AdjustmentListener, 340
 - associating components with, 341-342
 - FocusListener, 340
 - importing, 341
 - ItemListener, 340
 - KeyListener, 340
 - MouseListener, 340
 - MouseMotionListener, 340
 - WindowListener, 340
 - focus events, 346
 - item events, 349
 - key events, 350-351
 - methods, 342
 - actionPerformed(), 342
 - getSource(), 342
 - instanceof keyword, 343
 - mouse events, 351-352
 - mouse-movement events, 352
 - SwingColorTest sample application, 357
 - window events, 357
 - Exception class, 190-191**
 - in catch blocks, 194
 - exception classes, constructors, 204**
 - @exception tag (javadoc), 648**
 - exceptions, 188, 191. See also errors**
 - ArrayIndexOutOfBoundsException, 190
 - avoiding, 206
 - catching, 190-192
 - finally statement, 195-196
 - try and catch blocks, 192-194
 - class hierarchy, 190
 - classes, 193
 - compiler errors, 206-207
 - consistency checking, 191-192
 - creating, 204
 - EOFException, 191, 433
 - Error class, 190
 - Exception class, 190-191
 - floating, 205
 - handling nested handlers, 204-205
 - hierarchy, 205
 - inheritance, 204
 - IOException, 191, 453
 - limitations, 206
 - MalformedURLException, 191
 - methods, 198
 - multiple, 205
 - NullPointerException, 190, 453
 - overview, 215
 - passing, 201-202
 - throws keyword, 201
 - runtime, 190
 - SecurityException, 453
 - StreamCorruptionException, 453
 - Throwable class, 190
 - throwing, 190, 198-199, 203
 - explicit exceptions, 200
 - implicit exceptions, 200
 - inheritance issues, 202
 - throws keyword, 199-200
 - when to use, 205-206
 - exclamation point (!), NOT operator, 58**
 - exclusive radio buttons, 267**
 - executeQuery() method, 505-506**
 - exit command (jdb), 653**
 - exiting**
 - frames, 253
 - loops, 114
 - explicit casts, 81**
 - explicit exceptions, 200**
 - exponential notation, literals, 48**
 - expressions, 51**
 - definition of, 38
 - dot notation, 70
 - grouping, 58
 - readability, improving, 59
 - return values, 38, 51
 - extending interfaces, 172**
 - extends keyword, 120, 172**
 - extensions, .class, 637**
- ## F
- false value (Boolean), 48**
 - fatal errors, troubleshooting, 192**
 - feeds, 524**
 - DomainEditor application, creating, 532-534
 - DomainWriter application, creating, 536-538
 - RssFilter application, creating, 538-540
 - RssStarter application, creating, 529-531
 - RssWriter application, creating, 528
 - Field class, 459**
 - file input streams**
 - creating, 422
 - reading, 422-424
 - read() method, 423
 - while loops, 423
 - File objects, 440**

- file output streams**
 - creating, 425
 - writing to, 426
 - sample application, 425-426
 - write() method, 425
 - file streams**
 - file input, 422-423
 - file output, 425-426
 - FileInputStream class, 485**
 - FileInputStream() method, 422**
 - FileOutputStream class, 485**
 - FileOutputStream() method, 425**
 - FileReader() method, 436**
 - files**
 - archiving, jar utility, 648-650
 - compiling, 639-640
 - file objects, creating, 440
 - formats, jar, 648-650
 - installation (SDK 1.4), 615
 - multiple, compiling, 639
 - renaming, 441
 - unformatted, saving, 626
 - files formats, jar, 648-650**
 - FileWriter() method, 439**
 - fills, Java2D, 378-379**
 - filtering streams, 421, 426**
 - byte filters, 427
 - FilterInputStream class, 426**
 - FilterOutputStream class, 426**
 - filters, associating, 421**
 - filters (streams), 421**
 - final abstract methods, 182**
 - final class, 159**
 - final classes, 160-161**
 - subclassing, 160
 - final keyword, 44**
 - final method, 159**
 - final methods, 160**
 - final modifier, 159**
 - final variable, 159**
 - final variables, 159**
 - finally statement, 195-196**
 - Find command (Start menu), 623**
 - finding current color, 377**
 - finger protocol, 474**
 - Finger.java application, 475-476**
 - FingerServer sample application, 492-495**
 - finishConnect() method, 491**
 - first() method, 517**
 - flags, 223-228**
 - flip() method, 483**
 - float data type, 42**
 - floating exceptions, 205**
 - floating-point literals versus hexadecimal numbers, 376**
 - floating-point numbers, 42, 387**
 - exponents, 48
 - representing as literals, 48
 - flow layout manager, 307**
 - FlowLayout() constructor, 309
 - FlowLayout class, 306-307**
 - flowLayout() method, 307**
 - arguments, 309
 - focus, object-oriented programming, 14**
 - focus events, event handling, 346**
 - focusGained() method, 346**
 - FocusListener event listener, 340**
 - methods, 346
 - focusLost() method, 346**
 - folder structure (packages), 166**
 - folders, MS-DOS**
 - creating, 620
 - opening, 618-619
 - Font objects, creating, 370-372**
 - Font styles, selecting, 371**
 - FontMetrics class, 373-375**
 - fonts**
 - antialiasing, 372
 - built-in, 371
 - Font objects, creating, 370-372
 - returning information about, 373-375
 - for loops, 108-109, 231-233**
 - empty statements, 109
 - example, 110
 - troubleshooting, 109
 - Form.java, 263-264**
 - format, commands (JDK), 636**
 - formatting XML documents, 536-538**
 - forms, submitting (POST method), 496**
 - forName() method, 458, 484**
 - forward slash (/), comment notation, 46**
 - frames**
 - displaying, 252
 - exiting, 253
 - locations, 252
 - sizing, 251
 - user interfaces, 255
 - visible, 252
 - frameworks, developing, 254-255**
 - functionality, classes, 25**
 - functions, tools (modifying), 636. See also methods**
- G**
- g option (debugger), 651**
 - garbage collection, 69**
 - general utility methods, 76**
 - GeneralPath objects, creating, 383**
 - generics, 241-244**
 - get() method, 229, 236, 481**
 - getActionCommand() method, 345**

- getAppletInfo() method, 642-643, 653
- getChannel() method, 485
- getClass() method, 87
- getClickCount() method, 352
- getColor() method, 377
- getConnection() method, 504-505
- getConstructors() method, 460
- getContent() method, 468
- getContentType() method, 472
- getDate() method, 506
- getDouble() method, 506
- getFloat() method, 506
- getFontMetrics() method, 373
- getHeaderField() method, 472
- getHeaderFieldKey() method, 472
- getHeight() method, 373
- getInt() method, 506
- getItemAt() method, 269
- getItemCount() method, 269
- getKeyChar() method, 351
- getLong() method, 506
- getMessage() method, 193
- getModifiers() method, 460
- getName() method, 458
- getNumberOfFiles() method, 293
- getParameterInfo() method, 642-643
- getParameterTypes() method, 460
- getPoint() method, 352
- getProperties() method, 656
- getProperty() method, 656
- getResponseCode() method, 472
- getResponseMessage() method, 472
- getReturnType() method, 460
- getSelectedItem() method, 269
- getSelectedIndex() method, 269
- getSize() method, 258, 375
- getSource() method, 342, 345
- getStateChange() method, 349
- getString() method, 506
- getX() method, 352
- getY() method, 352
- Giftshop class, 177-178
- Gosling, James, 10
- gradient fills, 378
- graphical Swing applications, 251
- graphics
 - arcs, drawing, 382-383
 - coordinate spaces, 377-378
 - coordinate system, 369
 - drawing text, 370-372
 - antialiasing, 372
 - finding font metrics, 372-375
 - ellipses, drawing, 381
 - lines, drawing, 381
 - Map example, 384-387
 - polygons, drawing, 383-384
 - rectangles, drawing, 381
 - rendering attributes, 378
 - drawing strokes, 380
 - fill patterns, 378-379
- graphics operations, Java 2D, 82
- Graphics2D class, 368-369
- greater than or equal to symbol (\geq), 56
- greater than symbol ($>$), 56
- grid bag layout manager
 - creating, 327
 - versus grid layout manager, 325
- grids, designing, 327
- overview, 325
- grid bag layouts, creating, 325
- grid layout manager, 311
 - creating, 312
 - example, 312
 - versus grid bag layout manager, 325
- GridLayout() method, 312
- grids
 - creating grid layout manager, 312
 - designing, 327
 - element arrays, 97
- grouping
 - arguments, 131
 - classes, 32
 - expressions, 58
 - interfaces, 32
 - methods, 77
- GUIs
 - for Android apps, designing, 582-584
 - dialog boxes, 278
 - confirm dialog boxes, 279
 - Info sample application, 283-284
 - input dialog boxes, 280-281
 - message dialog boxes, 281
 - option dialog boxes, 282-283
 - event handling
 - action events, 345-346
 - ChangeTitle sample application, 343-345
 - component setup, 341-342
 - event listeners, 340
 - focus events, 346
 - item events, 349-350
 - key events, 350-351
 - methods, 342-343
 - mouse events, 351-352
 - mouse-movement events, 352
 - SwingColorTest sample application, 357
 - window events, 357
 - layout managers, 306
 - border layout, 314-315
 - card layout, 317-318

- combining, 316-317
- flow layout, 307-309
- grid bag layout, 325-327
- grid layout, 311-312
- insets, 334
- specifying, 306-307
- progress bars, 293
 - labels, 293
 - orientation, 293
 - updating, 293
- scroll panes, 288
 - creating, 288
 - scrollbars, 289
 - sizing, 288
 - toolbars, 289-290
- sliders, 286
 - advantages, 286
 - labels, 286-287
 - orientation, 286
 - Slider.java sample application, 287-288
- windows, resizing, 306

H

- HalfDollars application**
 - main() method, 97
 - source code, 95
- HalfDollars.java application (arrays), 96**
- HalfLoop.java application, 110**
- handlers, 555-560**
 - nested, 205
- handling**
 - arguments in applications, 131-132
 - strings, 79
- Harold, Elliott Rusty, 527**
- hash codes, 238**
- hash maps, 223, 236-241**
 - clearing, 237
 - creating, 237

- hash codes, 238
- load factor, 237
- hashCode() method, 238**
- HashMap class, 223, 236-241**
- hasNext() method, 224, 491**
- helper classes, 130**
- hexadecimal numbers, 47**
 - versus floating-point literals, 376
- HexRead.java application, 196**
- hierarchies**
 - classes, 26
 - creating, 27-29
 - defining, 161
 - designing, 28-29
 - exceptions, 205
 - interface, 172
 - methods, 30
- history**
 - of Android, 566-567
 - of Java, 10
- HolidaySked class, 226-227**
- HTML, forms (submitting), 496**
- HTML documents, viewing, 641**
- URLConnection class, 472**
- URLConnection objects, 468**
- hyphen (-), decrement operator (—), 55-56**

I

- I/O (input/output), 419**
 - nonblocking I/O, 488-495
 - streams, 419-420
 - buffered, 427-430
 - byte streams, 420, 426
 - character streams, 420, 436-440
 - creating, 420
 - data streams, 432-435
 - filtering, 421, 426-427
 - filters, 421
 - reading, 420
 - writing to, 421
- icons, 259-261**
 - JNLP files, 407
- Icons.java application, 259-260**
- identifying classes, 162**
- if keyword, 99**
- if statements, 99**
 - else keyword, 99
 - examples, 100
 - nested, 101
- ignore command (jdb), 654**
- ImageIcon objects, 259**
 - creating, 259
- implementing**
 - arrays, 92
 - interfaces, 168-169
 - multiple, 169
- implements keyword, 168-169, 208**
- implicit exceptions, 200**
- import declaration, 163-164**
- import statement, 167, 250**
- importing**
 - classes, 163
 - event listeners, 341
 - packages, 163-164
- improving readability**
 - expressions, 59
 - programs, 45
- increasing efficiency, classes, 160**
- increment operations, 54**
- increment operator (++), 55-56**
- incrementing variables, 54**
- increments, loops, 108**
- indexOf() method, 74, 231**
- inequality operator (!=), 85**
- InetAddress class, 489**
- Info application (dialog box example), 283-284**

- Info command (Applet menu), 642**
- Info.java application, 283-284**
- inheritance, 25**
 - access control, 155-156
 - class hierarchies, creating, 27-29
 - classes, 29-30
 - object-oriented programming, 25-26
 - exceptions, 202-204
 - multiple, 167
 - object-oriented programming, 27
 - overview, 27
 - single, 31, 167-168
- initialization, loops, 108**
- initializing objects, 69**
- inner classes, 179-180, 359-361, 388**
 - advantages, 181
 - naming, 181
 - scope, 181
- input dialog boxes, 280-281**
- input streams, 371, 452. See also streams**
 - creating, 453
 - ObjectReader class, 454-455
 - reading, 453-454
 - transient variables, 455-456
 - validation checks, 456-457
- input/output. See I/O (input/output)**
- InputDialog dialog boxes, 278**
- InputStream class, 422**
- insets, 334**
- inspecting classes, 457-463**
- installation files (SDK 1.4), 615**
- installing**
 - Android plug-in for Eclipse, 608-611
 - Android SDK, 607
 - Eclipse, 606
 - NetBeans, 596
 - SDK 1.4, 615-617
- instance methods. See methods**
- instance variables, 18, 39, 70**
 - versus class variables, 72
- defining, 120-121**
- length, 95**
- modifying, 71**
- testing, 71**
- values**
 - accessing, 70
 - initial values, 40
 - modifying, 70-72
- InstanceCounter class, 158-159**
- instanceof keyword, 343**
- instanceof operator, 58, 87**
- instances, 66. See also objects**
 - comparing, 85
 - of classes, creating, 16
 - passing, 81
- int data type, 42**
- integer literals, 47**
 - floating-point numbers, 48
 - negative numbers, 47
 - octal numbers, 47
- integers, data types, 42**
- interface hierarchy, 172**
- interfaces, 31-32**
 - as variable type, 169
 - casting objects to, 170
 - command-line, 617-618
 - creating, 170-171, 251-253
 - declaring, 168-171
 - Enumeration, 223
 - event listeners. *See* event listeners
 - extending, 172
 - grouping, 32
 - implementing, 168-169
 - Iterator, 223-225, 231
 - Map, 235-236
 - accessing elements, 235
 - methods, 171-172
 - multiple, 169
 - objects to, casting, 82
 - overview, 32, 167
 - protection, 171
 - Runnable, 208
 - Serializable, 448
 - variables, 171
 - versus classes, 168
- interpreter (java), 637-639**
- interpreters**
 - running, 621
 - SDK 1.4, 629
- InterruptedException exceptions, 192**
- introspection. See reflection**
- invoking, 73. See also calling**
- IOException, 191, 453**
- isAcceptable() method, 491**
- isConnectible() method, 491**
- isConnectionPending() method, 491**
- isEmpty() method, 236**
- ISO-8859-1 character encoding, 484**
- ISO-LATIN-1 character encoding, 484**
- isReadable() method, 491**
- isWritable() method, 491**
- Item class, 173-174**
- item events, event handling, 349**
- ItemListener event listener, 340**
- ItemProp application, 656**
- itemStateChanged() method, 349**
- Iterator interface, 223-225, 231**
- iterator() method, 231**

J**J2SE 1.3, 9****Jakarta, 182****jar utility, 648-650****Java**

- applications, 130
 - creating, 129-130
 - running, 637
- case-sensitivity, 41
- code, complexity, 206
- compilers, 614
- development tools, selecting, 614
- fonts, built-in, 371
- frames, exiting, 253
- XML, processing, 526

Java 2 Class Library, 16**Java 2 SDK 1.3, 9****Java 2D graphics operations, 82****Java data structures, 222****Java Database Connectivity. See JDBC****Java Development Kit. See JDK****Java exceptions. See exceptions****java interpreter, 637-639****Java Plug-in, 641****Java programs**

- compiling in Windows, 628-630
- developing, 614
- running in Windows, 628-630
- speed, 640

Java Web site, 636**Java Web Start, 391-395****JNLP files**

- creating, 396-405
- descriptions, 406
- icons, 407
- security, 406

- security privileges, selecting, 394
- server support, 405-406
- java.io package, 419. See also streams**
- java.lang package, 83**
- java.lang.reflect package, 459-461**
- java.net package. See networking**
- java.nio package, 481**
 - buffers, 481-483
 - byte buffers, 483-484
- java.nio.channels package, 485-488**
 - BufferConverter sample application, 486-488
 - nonblocking I/O, 488-495
- java.nio.charset package, 484-485**
- java.sql package, 501. See also JDBC (Java Database Connectivity)**
- java.util package, 151. See also data structures**
- java.vendor system property, 655**
- java.version system property, 655**
- Java2D**
 - arcs, drawing, 382-383
 - coordinate spaces, 377-378
 - ellipses, drawing, 381
 - lines, drawing, 381
 - polygons, drawing, 383-384
 - rectangles, drawing, 381
 - rendering attributes, 378
 - drawing strokes, 380
 - fill patterns, 378-379
 - sample applet, 384-387
- javac**
 - options
 - depreciation, 640
 - ea, 639
 - O, 640

- q, 640
- verbose, 640

- running Windows platforms, 628

javac command-line tool, 628**javac compiler, 639-640****javadoc, tags, 645**

- @author, 645
- @deprecated, 648
- @exception, 648
- @param, 648
- @return, 645
- @see, 648
- @serial, 645
- @since, 648
- @version, 645

javadoc documentation tool, 644-648

- AppInfo2 sample application, 645-647

- tags, 645, 648

javax.swing package, 151**javax.swing.JButton class, 17****javax.swing.JComponent class, 258****JCheckBox class, 266****JCheckBox() constructor, 267****JComboBox class, 269****jdb debugger, 650-654**

- applet debugging, 653
- application debugging, 651-653
- breakpoints, 651
 - deleting, 652-653
 - setting, 652
- commands
 - !!, 653
 - catch, 654
 - classes, 654
 - clear, 652
 - cont, 653
 - down, 654
 - exit, 653

- ignore, 654
- list, 653
- locals, 653
- memory, 654
- methods, 654
- print, 653
- resume, 654
- run, 652
- step, 653
- stop at, 652
- stop in, 652
- suspend, 654
- threads, 654
- up, 654
- single-step execution, 651
- JDBC (Java Database Connectivity), 500-501**
 - databases, accessing, 501
 - drivers, 501
 - ODBC data source connections
 - closing, 507
 - opening, 503-507
- JDK (Java Development Kit), 635-636**
 - command line, arguments, 636
 - commands, format, 636
 - downloading, 636
 - system properties, 655-656
 - utilities
 - applet viewer browser, 640-644
 - command line, 636
 - jar, 648-650
 - java interpreter, 637-639
 - javac compiler, 639-640
 - javadoc documentation tool, 644-648
 - jdb debugger, 650-654
 - options, 636-637
- JDK 1.3, 9**

JFC

- JOptionPane, 278**
 - confirm dialog boxes, 279
 - input dialog boxes, 280-281
 - message dialog boxes, 281
 - option dialog boxes, 282-283
- JProgressBar, 293**
- JScrollPane, 288-289**
- JSlider, 286-287**
- JToolBar, 289-290**
- JFrame() constructor, 251**
- JLabel() methods, 261**
- JNLP files**
 - creating, 396-405
 - descriptions, 406
 - icons, 407
 - security, 406
- JOptionPane class, 278**
 - confirm dialog boxes, 279
 - input dialog boxes, 280-281
 - message dialog boxes, 281
 - option dialog boxes, 282-283

JPanel class, 368**JProgressBar class, 293****JPython language, 638****JRadioButton class, 266****JRuby language, 638****JScrollPane class, 288-289****JScrollPane() method, 265****JSlider class, 286-287****JTextArea() method, 263****TextField() method, 262****JToolBar class, 289-290****JudoScript language, 638****junction styles (drawing strokes), 380****K****key-mapped data structures**

- Dictionary class, 223
- HashMap class, 223, 236-241
- Map interface, 235-236

keyboard, event handling, 351**KeyListener, 351****KeyListener event listener, 340****keyPressed() method, 351****keyReleased() method, 351****keystores, 405****keyTyped() method, 351****keywords**

- abstract, 161
- break, 114
- class, 120
- continue, 114
- else, 99
- extends, 120, 172
- final, 44, 159
- if, 99
- implements, 168-169, 208
- instance of, 343
- modifiers. *See* modifiers
- null, 93
- private, 152-153
- protected, 154
- public, 154
- return, 123
- static, 72, 120-121, 128, 157
- super, 142
- this, 124, 138
- throws, 199-201

L**labeled loops, 114****labels, 261-262**

- aligning, 261
- progress bars, 293
- sliders, 286-287

- languages**
 - JPython, 638
 - JRuby, 638
 - JudoScript, 638
 - NetRexx, 638
 - SQL (Structured Query Language), 500-501
- last() method, 517**
- layout managers, 306**
 - border layout, 315-316
 - creating, 314-315
 - card layout, 317
 - creating, 318
 - combining, 316-317
 - flow layout, 307-308
 - FlowLayout() constructor, 309
 - grid bag layout
 - creating, 327
 - designing grids, 327
 - overview, 325
 - grid layout, 311-313
 - creating, 312
 - example, 312
 - insets, 334
 - specifying, 306-307
- layouts, creating grid bag, 325**
- length instance variable, 95**
- length() method, 74**
- less than or equal to symbol (=), 56**
- less than symbol (), 56**
- lexical scope, 115**
- GPL (Lesser General Public License), 528**
- Line2D.Float class, 381**
- lines, drawing (Line2D.Float class), 381**
- lineTo() method, 383**
- linking node objects, 88**
- list command (jdb), 653**
- listeners, 340**
 - ActionListener, 340
 - AdjustmentListener, 340
 - associating components with, 341-342
 - FocusListener, 340
 - methods, 346
 - importing, 341
 - ItemListener, 340, 349
 - itemStateChanged() method, 349
 - KeyListener, 340, 351
 - MouseListener, 340, 352
 - MouseMotionListener, 340
 - WindowListener, 340, 357
- listings**
 - AllCapsDemo.java application, 441-442
 - Alphabet.java, 308
 - arrays, HalfDollars.java application, 96
 - BufferConverter class, 487
 - Bunch.java, 312-313
 - Buttons.java, 256-257
 - ChooseTeam.java, 268-270
 - CoalTotals.java, 508
 - CodeKeeper class, 232
 - CodeKeeper2 class, 243-244
 - ComicBooks class, 239-240
 - comparing objects, 85-86
 - defining methods, 123
 - DigitalClock.java, 209
 - Ellsworth example source code, 627
 - event handling
 - ChangeTitle.java application, 343-344
 - SelectItem application, 349
 - Finger.java application, 475
 - FingerServer class, 492-494
 - for loops, 110
 - Form.java, 263-264
 - Giftshop class, 177-178
 - HolidaySked class, 226-227
 - Icons.java, 259-260
 - Info.java application, 283-284
 - instance variables, testing/modifying, 71
 - InstanceCounter class, 158
 - Item class, 173-174
 - layout managers, border layout, 315-316
 - loops, while, 112
 - Map class, 384-386
 - MethodInspector class, 461
 - methods, calling, 73
 - MyRect class definition, 135
 - MyRect2 class, 139
 - NamedPoint class, 144
 - ObjectReader class, 454-455
 - objects, creating, 67-68
 - ObjectWriter class, 451-452
 - PassbyReference class, 126
 - PrintClass class, 141
 - ReferencesTest.java application, 78
 - SelectItem.java application, 349-350
 - simple arithmetic example, 52-53
 - SimpleFrame.java, 254
 - Slider.java application, 287
 - SquareTool class, 180
 - statements, switch, 103-104
 - Storefront class, 176-177
 - streams
 - BufferDemo.java application, 428-429
 - ReadBytes.java application, 424
 - ReadPrimes.java application, 435
 - ReadSource application, 438
 - WriteBytes application, 425-426
 - WritePrimes.java application, 434

SumAverage.java application, 132
 TextFrame class, 373-374
 TimeServer class, 478-479
 ToolBar.java application, 290
 try...finally blocks, 196
 VolcanoRobot class example, 21
 WebReader class, 469-470

lists, 271**literals, 46**

Boolean, 48-49
 characters, 49
 escape codes, 49-50
 integers, 47
 floating-point numbers, 48
 negative numbers, 47
 octal numbers, 47
 strings, 50-51

load factor (hash maps), 237**local variables, 39****locals command (jdb), 653****locations, frames, 252****logical operators, 57-58****long data type, 42****looping through array lists, 231-233****loops**

breaking, 114
 do...while, 113
 for, 108-109
 empty statements, 109
 example, 110
 troubleshooting, 109
 increments, 108
 index values, 109
 initialization, 108
 labeling, 114
 restarting, 114
 run() methods, 214
 tests, 108
 while, 111-113
 overview, 111

lowercase, converting to uppercase (AllCapsDemo application), 441**M****main() method, 24, 129, 154, 637**

signature, 129

makeRange() method, 123-124**MalformedURLException exception, 191****management, memory, 69****managing errors/exceptions. See error-handling****manifest file, configuring for Android apps, 581-582****Map applet, 384-387****Map class, 384-386****Map interface, 235-236**

elements, accessing, 235

maps, hash maps, 223, 236-241

clearing, 237
 creating, 237
 hash codes, 238
 load factor, 237

math operators, 51-52

example, 52-53

MD command (MS-DOS), 620**member functions. See methods****memory, 69****memory command (jdb), 654****memory management, 69****menus, 295-296, 299****message dialog boxes, 281****MessageDialog dialog boxes, 278****method calls, nesting, 75-76****Method class, 459-460****method overloading, 122****MethodInspector class, 461****methods**

abstract methods, 161-162
 final, 182
 private, 182
 access control, 151, 155
 default access, 151
 inheritance, 155-156
 private access, 152-153
 protected access, 154-155
 public access, 153-154
 accessor methods, 156
 actionPerformed(), 342
 add(), 229, 256
 addActionListener(), 345
 addItem(), 269
 afterLast(), 517
 allocate(), 482
 ArgStream(), 430
 arguments, passing to, 126-127
 beforeFirst(), 517
 BorderLayout(), 314
 BufferedInputStream(), 427
 BufferedOutputStream(), 428
 BufferedReader(), 437
 BufferedWriter(), 440
 buildRect(), declaring, 134
 calling, 19, 73-74
 CardLayout(), 318
 channel(), 491
 charAt(), 74
 charWidth(), 373
 checkTemperature(), 25
 class, 19
 accessing, 157
 class methods, 76-77, 127
 availability, 128
 calling, 77
 defining, 128
 classes, 19
 clear(), 230, 237
 close(), 421

- closePath(), 384
- configureBlocking(), 490
- connect(), 490
- constructors, 67, 136-138
 - calling, 136, 143
 - calling from another constructor, 138
 - definition of, 69
 - naming, 137
 - overloading, 138-139
 - overriding, 143-144
- contains(), 230
- containsKey(), 237
- containsValue(), 237
- createFont(), 371
- createStatement(), 505
- DataInputStream(), 432
- DataOutputStream(), 433
- decode(), 485
- defining, 122
 - parameter lists, 122
 - this keyword, 124
- definitions, multiple, 133
- delete(), 441
- deriveFont(), 372
- drawString(), 370
- empty(), 235
- encode(), 485
- equals(), 85, 238
- event handling, 342
- exceptions, 198
- executeQuery(), 505-506
- FileInputStream(), 422
- FileOutputStream(), 425
- FileReader(), 436
- FileWriter(), 439
- final, 159
- final methods, 160
- finishConnect(), 491
- first(), 517
- flip(), 483
- flowLayout(), 307
 - arguments, 309
- focusGained(), 346
- focusLost(), 346
- forName(), 458, 484
- general utility, 76
- get(), 229, 236, 481
- getActionCommand(), 345
- getAppletInfo(), 642-643, 653
- getChannel(), 485
- getClass(), 87
- getClickCount(), 352
- getColor(), 377
- getConnection(), 504-505
- getConstructors(), 460
- getContent(), 468
- getContentType(), 472
- getDate(), 506
- getDouble(), 506
- getFloat(), 506
- getFontMetrics(), 373
- getHeaderField(), 472
- getHeaderFieldKey(), 472
- getHeight(), 373
- getInt(), 506
- getItemAt(), 269
- getItemCount(), 269
- getKeyChar(), 351
- getLong(), 506
- getMessage(), 193
- getModifiers(), 460
- getName(), 458
- getNumberOfFiles(), 293
- getParameterInfo(), 642-643
- getParameterTypes(), 460
- getPoint(), 352
- getProperties(), 656
- getProperty(), 656
- getResponseCode(), 472
- getResponseMessage(), 472
- getReturnType(), 460
- getSelectedIndex(), 269
- getSelectedItem(), 269
- getSize(), 258, 375
- getSource(), 342, 345
- getStateChange(), 349
- getString(), 506
- getX(), 352
- getY(), 352
- GridLayout(), 312
- grouping, 77
- hashCode(), 238
- hasNext(), 224, 491
- hierarchies, 30
- indexOf(), 74, 231
- instance methods, 128
- interfaces, 31, 171-172
- isAcceptable(), 491
- isConnectible(), 491
- isConnectionPending(), 491
- isEmpty(), 236
- isReadable(), 491
- isWritable(), 491
- itemStateChanged(), 349
- iterator(), 231
- JLabel(), 261
- JScrollPane(), 265
- JTextArea(), 263
- JTextField(), 262
- keyPressed(), 351
- keyReleased(), 351
- keyTyped(), 351
- last(), 517
- length(), 74
- lineTo(), 383
- main(), 24, 129, 154, 637
 - signature, 129
- makeRange(), 123-124
- mouseClicked(), 352
- mouseDragged(), 352
- mouseEntered(), 352
- mouseExited(), 352
- mouseMoved(), 352
- mousePressed(), 352
- mouseReleased(), 352

- moveTo(), 383
- nested, combining, 76
- newDecoder(), 485
- newEncoder(), 485
- newInstance(), 459
- newLine(), 440
- next(), 224, 491
- of objects, calling, 76
- overloaded
 - creating, 133-136
 - definition of, 133
 - troubleshooting, 133
- overloading advantages, 133
- overriding, 30, 140-142
 - advantages, 142
 - super keyword, 142
- paintComponent(), 368-370, 377
- peek(), 235
- pop(), 234
- position(), 482, 485
- POST, form submission, 496
- previous(), 517
- println(), 45
- protecting, 162
- push(), 234
- put(), 236, 482
- putChar(), 483
- putDouble(), 483
- putFloat(), 483
- putInt(), 483
- putLong(), 483
- putShort(), 483
- read(), 420-421, 437, 454, 486
- readBoolean(), 454
- readByte(), 454
- readChar(), 454
- readDouble(), 454
- readFloat(), 433, 454
- readInt(), 454
- readLine(), 438, 454
- readLong(), 454
- readObject(), 453, 456
- readShort(), 454
- readStream(), 430
- readUnsignedByte(), 433, 454
- readUnsignedShort(), 433, 454
- register(), 490
- remove(), 229, 236
- renameTo(), 441
- return types, 122
 - void, 123
- run(), 209
 - loops, 214
- search(), 235
- select(), 491
- selectedKeys(), 491
- set(), 230
- setActionCommand(), 346
- setBackground(), 377
- setBounds(), 252
- setColor(), 376
- setDefaultCloseOperation(), 253
- setEchoChar(), 263
- setEnabled(), 258
- setFollowRedirects(), 472
- setFont(), 371
- setLayout(), 306, 317-318
- setLineWrap(), 263
- setMajorTickSpacing(), 286
- setMaximum(), 293
- setMaximumRowCount(), 269
- setMinimum(), 293
- setMinorTickSpacing(), 286
- setPaint(), 378
- setPaintLabels(), 287
- setPaintTicks(), 287
- setRenderingHint(), 372
- setSelected(), 266
- setSelectedIndex(), 269
- setSize(), 251, 258, 369
- setSoTimeout(), 473
- setStringPainted(), 293
- setStroke(), 380
- setValue(), 293
- setVisible(), 252, 258
- setWrapStyleWord(), 263
- show(), 318
- showConfirmDialog(), 279
- showInputDialog(), 280
- showMessageDialog(), 281
- showOptionDialog(), 282
- signatures, 122, 133
- SimpleFrame(), 255
- size, 231
- size(), 236, 486
- start(), calling, 641
- static, 157-159
- stop(), 214
 - calling, 641
- stringWidth(), 373
- substring(), 74
- in superclasses, calling, 143
- System.out.println(), 45
- testing, 124
- toUpperCase(), 74
- trimToSize(), 231
- URL(), 469
- valueOf(), 77
- windowActivated(), 357
- windowClosed(), 357
- windowClosing(), 357
- windowDeactivated(), 357
- windowDeiconified(), 357
- windowIconified, 357
- windowOpened(), 357
- wrap(), 481
- write(), 421, 428, 439, 450
- writeBoolean(), 450
- writeByte(), 450

- writeBytes(), 450
 - writeChar(), 450
 - writeChars(), 450
 - writeDouble(), 450
 - writeFloat(), 450
 - writeInt(), 435, 450
 - writeLong(), 450
 - writeObject(), 450
 - writeShort(), 450
 - writeStream(), 430
 - methods command (jdb), 654**
 - minus sign (-), 47**
 - Modifier class, 459**
 - modifiers, 150**
 - abstract, 161
 - final, 159
 - multiple, 150
 - private, 152-153
 - protected, 154
 - public, 154
 - static, 157
 - modifying. See also editing**
 - arrays, 95
 - class variable values, 72
 - classes, 208
 - functions, tools, 636
 - instance variable values, 70-72
 - instance variables, 71
 - precedence, operators, 59
 - superclasses, 27
 - modulus operators, 52**
 - mouse, event handling, 351-352**
 - mouseClicked() method, 352**
 - mouseEntered() method, 352**
 - mouseExited() method, 352**
 - MouseListener, 352**
 - MouseListener event listener, 340**
 - MouseMotionListener event listener, 340**
 - mouseMoved() method, 352**
 - mousePressed() method, 352**
 - mouseReleased() method, 352**
 - moveTo() method, 383**
 - MS-DOS**
 - accessing, 618
 - CLASSPATH variable
 - Windows 2000/XP, 632-633
 - Windows 9x/Me, 630-631
 - commands, cd, 628
 - folders
 - creating, 620
 - opening, 618-619
 - PATH variable
 - Windows 2000/XP, 623-625
 - Windows 9x/Me, 622-623
 - programs, running, 620
 - MS-DOS commands**
 - CD, 619
 - MD, 620
 - SET CLASSPATH=, 631
 - MS-DOS prompt, 636**
 - MS-DOS Prompt window, 618**
 - multidimensional arrays, 97**
 - multiple, method definitions, 133**
 - multiple bytes, writing, 425**
 - multiple exceptions, 205**
 - multiple files, compiling, 639**
 - multiple inheritance, 167**
 - multiple interfaces, 169**
 - multiple modifiers, 150**
 - multitasking, 207**
 - MyRect class, 133**
 - buildRect() method, 134
 - MyRect class definition, 135**
 - MyRect2 class, 139**
- N**
- name conflicts (classes), 165**
 - NamedPoint class, 144**
 - names, reducing conflicts, 162**
 - namespaces, 529**
 - naming**
 - classes, inner, 181
 - methods, constructor, 137
 - packages, 165-166
 - variables, 40-41
 - naming conventions, 165**
 - Native Interface Header Files component (SDK 1.4), 616**
 - navigating records, 507, 517**
 - Naviseek Web site, 24**
 - negative numbers, representing as literals, 47**
 - nested if statements, 101**
 - nested methods, combining, 76**
 - nesting**
 - exception handlers, 204-205
 - method calls, 75-76
 - NetBeans**
 - installing, 596
 - Java class, creating, 598-599
 - projects, creating, 596
 - NetRexx language, 638**
 - networking, 468**
 - channels, 485-488
 - BufferConverter sample application, 486-488
 - non-blocking I/O, 488-495
 - character sets, 484-485
 - creating, 484
 - java.nio package, 481
 - buffers, 481-484
 - sockets, 473
 - client-side, 473-474
 - Finger.java application, 475-476

- server-side, 476-477
 - transport-layer implementation, 477
 - streams, 468
 - BufferedReader objects, 468
 - getContent() method, 468
 - URLConnection objects, 468
 - URL objects, 468
 - WebReader sample application, 469-473
 - TimeServer application
 - designing, 477-479
 - testing, 479-480
 - new operator, 59, 66-69**
 - instantiating arrays, 93
 - objects, creating with, 66-69
 - newDecoder() method, 485**
 - newEncoder() method, 485**
 - newInstance() method, 459**
 - newline() method, 440**
 - next() method, 224, 491**
 - NoClassDef error, 630**
 - node objects, linking, 88**
 - nonblocking I/O, over networking channels, 488-495**
 - nonexclusive check boxes, 267**
 - not equal symbol (!=), 56**
 - NOT operator, 58**
 - notation, dot, 70**
 - Notepad, accessing, 622**
 - null keyword, 93**
 - NullPointerException, 453**
 - NullPointerException exception, 190**
 - number literals, 47**
 - floating-point numbers, 48
 - negative numbers, 47
 - octal numbers, 47
 - numbering systems, 47**
 - numbers**
 - floating-point, 42
 - octal, 47
- O**
- O option (debugger), 651**
 - Object class, 26**
 - object serialization, private variables and objects, 464**
 - object streams, 448. See also streams**
 - Writer and Reader classes, 463
 - object variables. See instance variables**
 - object-oriented programming (OOP), 13**
 - classes, 14-15
 - inheritance, 25-26
 - focus, 14
 - inheritance, 27
 - class hierarchies, 27-29
 - single, 31
 - objects, 14
 - ObjectInputStream class, 452-455**
 - ObjectOutputStream class, 449-451**
 - ObjectReader class, 454-455**
 - objects, 13-14. See also instances**
 - arrays, creating, 93-94
 - attributes, defining, 18
 - ButtonGroup, 267
 - casting, 79-80
 - to interfaces, 170
 - classes
 - attributes, 17-18
 - behavior, 18-19
 - casting, 81-82
 - determining, 87
 - Color, 375
 - comparing, 85-86
 - converting, 79-80
 - creating, 66-68
 - arguments, 66
 - constructors, 69
 - with new operator, 66-69
 - current, referring to, 124
 - encapsulating, 153
 - File, 440
 - Font, creating, 370-372
 - GeneralPath, creating, 383
 - ImageIcon, 259
 - initializing, 69
 - to interfaces, casting, 82
 - memory, 69
 - methods of, calling, 76
 - nodes, linking, 88
 - primitive types, converting, 83
 - references, 77, 85
 - example, 78-79
 - reusing, 15-16
 - Selector, 490
 - serialization, 448-449
 - advantages, 449
 - input streams, 452-455
 - output streams, 449-452
 - persistence, 449
 - transient variables, 455-456
 - validation checks, 456-457
 - String, creating, 21
 - StringTokenizer, creating, 67-68
 - URL, creating, 469
 - ObjectWriter class, 451-452**
 - obscuring password fields, 263**
 - octal numbering systems, 47**
 - octal numbers, 47**
 - ODBC, data source connections**
 - closing, 507
 - opening, 503-507
 - Old Native Interface Header Files component (SDK 1.4), 616**
 - online storefronts, creating, 172-179**
 - OOP. See object-oriented programming**

opening

- folders, MS-DOS, 618-619
- socket connections, 473
- streams over Internet, 468
 - BufferedReader objects, 468
 - getContent() method, 468
 - URLConnection objects, 468
 - URL objects, 468
 - WebReader sample application, 469-473

operations, 54**operators, 51**

- arithmetic, 51-52
 - example, 52-53
- assignment, 53-54
 - equal sign (=), 40
- comparison, 56-57
- concatenation (+), 60
- conditional, 107-108
- decrement (—), 55-56
- division, 52
- equality (==), 85
- increment (++), 55-56
- inequality (!=), 85
- instance of, 58, 87
- logical, 57-58
- modulus, 52
- new, 59, 66-69
 - creating objects with, 66-69
 - instantiating arrays, 93
- postfix, 55
- precedence, 58-59
 - modifying, 59
- prefix, 55
- subtraction, 52

option dialog boxes, 282-283**OptionDialog dialog boxes, 278****options, command line, 636-637****OR operators, 57****Oracle, 636****order of precedence, operators, 58-59**

- modifying, 59

organization

- classes, 150
- stacks, 233

organizing

- behavior, classes, 25-32
- classes, 25-32, 162
- projects, Android, 569-570

os.name system property, 655**os.version system property, 655****output streams, 449. See also streams**

- creating, 450
- ObjectWriter class, 451-452
- writing to, 450

overflow (variable assignment), 62**overloaded methods**

- buildRect(), 134
- constructors, 138-139
- creating, 133-136
- definition of, 133
- troubleshooting, 133

overloading methods, 122

- advantages, 133

overriding

- methods, 30, 140-142
 - advantages, 142
 - constructors, 143-144
 - super keyword, 142
- scrollbars, 289

P**package declaration, 152, 166****packages, 32, 162**

- access control, 166
- default access, 166
- public access, 166-167

- advantages, 162

- classes, creating, 638

- creating, 165-166

- importing, 163-164

- java.lang, 83

- java.lang.reflect, 459-461

- java.net package. *See* networking

- java.nio package, 481

- buffers, 481-484

- java.nio.channels package, 485-488

- BufferConverter sample application, 486-488

- nonblocking I/O, 488-495

- java.nio.charset package, 484-485

- java.util, 151

- javax.swing, 151

- overview, 32

- referencing, 162

- class names, 162

- package names, 163

paintComponent() method, 368-370, 377**panels, 256, 317**

- components, adding, 317

- creating, 317

- insets, 334

panes, scrolling, 265**parallel data, 448****@param tag (javadoc), 648****Pardon My Icons Web site, 261****PassByReference class, 126****passing**

- arguments

- to applications, 130

- to methods, 126-127

- exceptions, 201-202

- throws keyword, 201

- instances, 81

password fields, obscuring, 263

PATH variable (MS-DOS)

- Windows 2000/XP, 623-625

- Windows 9x/Me, 622-623

- peek() method, 235**

- performance, assessor methods, 182**

- period (.), dot notation, 70**

- persistence, 449**

- pipe character (|), OR operators, 57**

- plus sign (+)**

- concatenation operator (+), 60

- increment operator (++), 55-56

- pointers, 79. See also arrays; references**

- pointers (C/C++), 79, 88**

- polygons, drawing (Java2D), 383-384**

- pop() method, 234**

- port numbers, selecting, 477**

- position() method, 482, 485**

- POST method, form submission, 496**

- postfix operators, 55**

- potential exceptions, indicating, 199**

- precedence, operators, 58-59**

- modifying, 59

- prefix operators, 55**

- preparing resources for Android apps, 578-580**

- previous() method, 517**

- primitive data types, buffers, 481**

- primitive type arrays, 458**

- primitive types, 88**

- casting, 79-81

- converting, 79-80

- data, 42

- objects, converting, 83

- print command (jdb), 653**

- PrintClass class, 141**

- println() method, 45**

- private abstract methods, 182**

- private access, 152-153**

- private modifier, 152-153**

- private variables, object serialization, 464**

- procedural programming, 13**

- processing XML**

- with Java, 526

- with XOM, 526-528

- programming**

- object-oriented. *See* object oriented programming

- procedural, 13

- programs. See also applications**

- compiling, 617

- debugging, 626, 650

- Java. *See* Java programs

- MS-DOS, running, 620

- Readability, improving, 45

- running, 23, 617

- progress bars, 293**

- projects, organizing (Android), 569-570**

- properties, system properties, 655-656**

- protected access, 154-155**

- protected modifier, 154**

- protecting classes, methods, and variables, 162**

- protection, interfaces, 171**

- public access, 153-154**

- public modifier, 154**

- push() method, 234**

- put() method, 236, 482**

- putChar() method, 483**

- putDouble() method, 483**

- putFloat() method, 483**

- putInt() method, 483**

- putLong() method, 483**

- putShort() method, 483**

Q-R

- q option (javac), 640**

- queries, databases, 500**

- quotation marks in arguments, 131**

- radio buttons, 266-267**

- event handling

- action events, 345-346

- item events, 349-350

- exclusive, 267

- Random() class, enabling, 69**

- RangeClass.java application, 123**

- read() method, 420-421, 437, 454, 486**

- readability**

- expressions, improving, 59

- programs, improving, 45

- readBoolean() method, 454**

- readByte() method, 454**

- ReadBytes.java application, 424**

- readChar() method, 454**

- readDouble() method, 454**

- Reader class, 463**

- readFloat() method, 433, 454**

- reading**

- bytes from channels, 486

- input streams, 453-454

- from sockets, 473

- streams, 420

- buffered streams, 427

- character streams, 436-439

- data streams, 433

- file input streams, 422-423

- readInt() method, 454**

- readLine() method, 438, 454**

- readLong() method, 454**

- readObject() method, 453, 456**

- ReadPrimes.java application, 435
 - readShort() method, 454
 - ReadSource.java application, 438
 - readStream() method, 430
 - readUnsignedByte() method, 433, 454
 - readUnsignedShort() method, 433, 454
 - reclaiming memory, 69
 - records
 - navigating, 507, 517
 - SQL, writing to database, 509-516
 - Rectangle2D.Double class, 381
 - Rectangle2D.Float class, 381
 - rectangles, drawing (Rectangle2D.Float class), 381
 - reducing conflicts, names, 162
 - references
 - arrays, 95
 - example, 78-79
 - objects, 85
 - passing by, 126
 - references (objects), 77
 - ReferencesTest.java application, 78
 - referencing packages, 162
 - class names, 162
 - package names, 163
 - referring to objects, current, 124
 - reflection, 457, 461-463
 - java.lang.reflect package, 459-461
 - register() method, 490
 - Reload command (Applet menu), 641
 - remove() method, 229, 236
 - removing array list elements, 230
 - renameTo() method, 441
 - renaming files, 441
 - rendering attributes (Java2D), 378
 - drawing strokes, 380
 - fill patterns, 378-379
 - requests, XML-RPC
 - responding to, 549-550
 - sending, 548-549
 - reserved words, 150. *See also* modifiers
 - resizing
 - components, 258
 - windows, 306
 - responding to XML-RPC requests, 549-550
 - Restart command (Applet menu), 641
 - restarting loops, 114
 - resume command (jdb), 654
 - return keyword, 123
 - @return tag (javadoc), 645
 - return types, 122
 - methods, void, 123
 - return values, 38, 51
 - reusing objects, 15-16
 - RMI (remote method invocation), 546
 - RPCs, 545
 - RMI, 546
 - XMC-RPC, 546-547
 - requests, responding to, 549-550
 - requests, sending, 548-549
 - XML-RPC
 - implementation, selecting, 550-551
 - web service, creating, 555-560
 - web service, selecting, 552-555
 - RSS Advisory Board, 524
 - RSS feed application, creating, 528-531
 - RSS feeds
 - DomainEditor application, creating, 532, 535
 - DomainWriter application, creating, 536-538
 - RssFilter application, creating, 538-540
 - RssFilter application, creating, 538-540
 - run command (jdb), 652
 - Run command (Start menu), 615
 - run() method, 209
 - loops, 214
 - Runnable interface, 208
 - running
 - Android app, 575-576
 - applications, 637
 - bytecode, 637
 - interpreter, 621
 - Java programs in Windows, 628-630
 - programs, 23, 617
 - in MS-DOS, 620
 - threads, 209
 - runtime errors, 242
 - runtime exceptions, 190, 215
- S**
- Sams Teach Yourself Android Application Development in 24 Hours, 2nd Edition*, 589, 607
 - saving
 - files, formatted., 626
 - source code, 628
 - scope
 - inner classes, 181
 - lexical scope, 115
 - variables, 98, 125-126
 - troubleshooting, 125
 - scroll panes, 288-289

scrollbars

- configuring, 265
- overriding, 289

scrolling panes, 265**ScrollPaneConstants class, 265****SDK 1.3, 9, 12****SDK 1.4**

- components, 616
- configuring, 617
 - CLASSPATH variable, 630-633
 - command-line interface, 617-618
 - configuration errors, 622
 - MS-DOS folders, 618-619
 - PATH variable, 622-625
 - troubleshooting, 616
- installation files, 615
- installing, 615-617
 - on Windows, 615-617
 - troubleshooting, 615
- interpreter, 629

SDK Setup Wizard, 615**SDK utilities, jdb (commands), 653-654****search() method, 235****searching**

- array list elements, 230
- stack, 235

security, JNLP files, 406**SecurityException, 453****@see tag (javadoc), 648****select() method, 491****selectedKeys() method, 491****selecting**

- development tool, 12
- development tools, 614
- Font styles, 371
- port numbers, 477
- XML-RPC implementation, 550-551
- XML-RPC web service, 552-555

SelectionKey class, 490-491**SelectItem.java application, 349-350****Selector objects, 490****semicolon (;), statement termination character, 38****sending XML-RPC requests, 548-549****serial data, 448****@serial tag (javadoc), 645****Serializable interface, 448****serialization, 448-449**

- advantages, 449
- input streams, 452
 - creating, 453
 - ObjectReader class, 454-455
 - reading, 453-454
- output streams, 449
 - creating, 450
 - ObjectWriter class, 451-452
 - writing to, 450
- persistence, 449
- private variables and objects, 464
- transient variables, 455-456
- validation checks, 456-457

server-side sockets, 476-477**ServerSocket class, 476-477****SET CLASSPATH= command (MS-DOS), 631****set() method, 230****setActionCommand() method, 346****setBackground() method, 377****setBounds() method, 252****setColor() method, 376****setDefaultCloseOperation() method, 253****setEchoChar() method, 263****setEnabled() method, 258****setFollowRedirects() method, 472****setFont() method, 371****setLayout() method, 306, 317-318****setLineWrap() method, 263****setMajorTickSpacing() method, 286****setMaximum() method, 293****setMaximumRowCount() method, 269****setMinimum() method, 293****setMinorTickSpacing() method, 286****setPaint() method, 378****setPaintLabels() method, 287****setPaintTicks() method, 287****setRenderingHint() method, 372****setSelected() method, 266****setSelectedIndex() method, 269****setSize() method, 251, 258, 369****setSoTimeout() method, 473****setStringPainted() method, 293****setStroke() method, 380****setting**

- breakpoints, 652
- drawing colors, 376-377

setValue() method, 293**setVisible() method, 252, 258****setWrapStyleWord() method, 263****shapes, coordinate system, 369****shared behavior, 31-32****shared values, defining, 43****short data type, 42****show() method, 318****showConfirmDialog() method, 279****showInputDialog() method, 280****showMessageDialog() method, 281****showOptionDialog() method, 282****ShowTokens.java, 67-68**

- signatures (methods), 122, 133
- SimpleFrame() method, 255
- SimpleFrame.java, 254
- @since tag (javadoc), 648
- single inheritance, 31, 167-168
- single-step execution, 651
- sites (Web), Java 2 documentation, 46
- size, array lists, 228
- size() method, 231, 236, 486
- sizing
 - array lists, 231
 - components, 258
 - frames, 251
 - scroll panes, 288
- sliders, 286
 - advantages, 286
 - labels, 286-287
 - orientation, 286
 - Slider.java sample application, 287-288
- SOAP (Simple Object Access Protocol), 547
- Socket class, 473
- SocketImpl class, 477
- sockets, 473
 - client-side, 473-474
 - Finger.java application, 475-476
 - nonblocking clients/servers, 489-495
 - server-side, 476-477
 - timeout values, 473
 - TimeServer application
 - designing, 477-479
 - testing, 479-480
 - transport-layer implementation, 477
- software deployment, Java Web Start, 392-395
- sources (casting), 80
- source code
 - comments, 644
 - converting, 639
 - Ellsworth example, 627
 - saving, 628
- source files, creating, 627-629
- Spartacus application
 - creating, 599-600
 - debugging, 601
- specifying
 - class files, 638
 - layout managers, 306
- speed, Java programs, 640
- splash screens, 407
- SQL, records (writing to database), 509-516
- SQL (Structured Query Language), 500-501
 - records, writing to database, 509-516
- SquareTool class, 180
- sRGB color system, 375
- Stack class, 223, 233-235
- stack frames, 654
- stacks, 223, 233-235, 654
 - elements, 234
 - logical organization, 233
 - searching, 235
- Start command (Applet menu), 641
- Start menu, commands
 - Find, 623
 - Run, 615
- start() method, calling, 641
- statements, 38. *See also* modifiers
 - block statements, 38
 - blocks, 98-99
 - try...catch, 193-194
 - conditionals
 - conditional operator, 107-108
 - if, 99-100
 - switch, 101-107
 - empty, for loops, 109
 - expressions, 51
 - definition of, 38
 - return values, 38, 51
 - finally, 195-196
 - if, nested, 101
 - import, 163-164, 167, 250
 - loops
 - breaking, 114
 - do, 113
 - for, 108-110
 - index values, 109
 - labeling, 114
 - restarting, 114
 - while, 111-113
 - switch, 101
 - termination character, 38
- static keyword, 72, 120-121, 128
- static methods, 157-159
- static modifier, 157
- static variables, 72, 157-159
- step command (jdb), 653
- stop at command (jdb), 652
- Stop command (Applet menu), 641
- stop in command (jdb), 652
- stop() method, 214
 - calling, 641
- Storefront application, 172
- Storefront class, 176-177
- storefronts, creating online, 172-179
- storing data, 427
- StreamCorruptionException exceptions, 453
- streams, 419-420, 448
 - buffered, 427
 - creating, 427-428
 - reading, 427
 - writing to, 428-430

- byte streams, 420-422
 - file input streams, 422-423
 - file output streams, 425-426
- character streams, 420, 436
 - reading text files, 436-439
 - writing text files, 439-440
- creating, 420
- data streams, 432-435
 - creating, 432
 - reading, 433
 - sample applications, 434
- filtering, 421, 426
 - byte filters, 427
- filters, 421
- input, 371
- input streams, 452
 - creating, 453
 - ObjectReader class, 454-455
 - reading, 453-454
 - transient variables, 455-456
 - validation checks, 456-457
- object streams, Writer and Reader classes, 463
- opening over Internet, 468
 - BufferedReader objects, 468
 - getContent() method, 468
 - URLConnection objects, 468
 - URL objects, 468
 - WebReader sample application, 469-473
- output streams, 449
 - creating, 450
 - ObjectWriter class, 451-452
 - writing to, 450
- reading, 420
- writing to, 421
- string arithmetic, 60**
- string literals, 50-51**
- String objects, creating, 21**
- strings**
 - concatenating, 60
 - handling, 79
- StringTokenizer class, 67**
- StringTokenizer objects, creating, 67-68**
- stringWidth() method, 373**
- strokes (drawing), 380**
- Structured Query Language (SQL), 500-501**
- structures, data. See data structures**
- styles, Font, 371**
- subclasses, 25**
 - defining, 26
- subclassing, 27**
 - final classes, 160
- submitting forms, POST method, 496**
- subscripts (arrays), 94**
- substring() method, 74**
- subtraction operator, 52**
- SumAverage.java application, 132**
- super keyword, 142**
- super(), 143**
- superclasses, 25**
 - casting, 82
 - indicating, 120
 - methods in calling, 143
 - modifying, 27
- supporting Java Web Start on servers, 405-406**
- surfaces, drawing, 368-369**
- suspend command (jdb), 654**
- Swing**
 - applications
 - graphical, 251
 - Slider.java, 287
- components, 250, 258
 - check boxes, 266-267
 - combo boxes, 269-270
 - creating, 250, 255-256
 - disabled, 258
 - drop-down lists, 269-270
 - icons, 259-261
 - labels, 261-262
 - lists, 271
 - radio buttons, 266-267
 - resizing, 258
 - scrolling panes, 265
 - text areas, 263-265
 - text fields, 262-263
- containers, panels, 256
- dialog boxes
 - confirm dialog boxes, 279
 - creating, 278
 - input dialog boxes, 280-281
 - message dialog boxes, 281
 - option dialog boxes, 282-283
 - sample application, 283-284
- event-handling, 339
 - action events, 345-346
 - ChangeTitle.java, 343-345
 - component setup, 341-342
 - event listeners, 341
 - focus events, 346
 - instanceof keyword, 343
 - item events, 349
 - key events, 350-351
 - methods, 342
 - mouse events, 351-352
 - mouse-movement events, 352
 - SwingColorTest sample application, 357
 - window events, 357
- Info application, 283-284

- layout managers, 306
 - border layout, 314-315
 - card layout, 317-318
 - combining, 316-317
 - flow layout, 307-309
 - grid bag layout, 325-327
 - grid layout, 311-312
 - insets, 334
 - specifying, 306-307
 - progress bars, 293
 - scroll panes, 288-289
 - sliders, 286
 - advantages, 286
 - labels, 286-287
 - orientation, 286
 - Slider.java sample application, 287-288
 - SwingWorker class, 408-413
 - ToolBar.java application, 290
 - toolbars, 289-290
 - windows, resizing, 306
 - SwingColorTest application, 357**
 - SwingConstants class, 261, 286**
 - SwingWorker class, 408-413**
 - switch statements, 101-107**
 - limitations, 102
 - system properties, 655-656**
 - System.out.println() method, 45**
- T**
- tabbed panes, 299-300
 - Tag command (Applet menu), 642**
 - tags**
 - applet, archive attribute, 649
 - javadoc, 645, 648
 - @author, 645
 - @deprecated, 648
 - @exception, 648
 - @param, 648
 - @return, 645
 - @see, 648
 - @serial, 645
 - @since, 648
 - @version, 645
 - XML, 523
 - tasks, statements, 38**
 - TCP sockets, 473**
 - client-side, 473-474
 - server-side, 476-477
 - TimeServer application
 - designing, 477-479
 - testing, 479-480
 - transport-layer implementation, 477
 - telnet, enabling, 480**
 - telnet connections, 479-480**
 - terminating threads, 214**
 - ternary operator, 107-108**
 - testing**
 - apps, Android, 611
 - instance variables, 71
 - methods, 124
 - socket client/server application, 479-480
 - tests, loops, 108**
 - text**
 - ASCII, 627
 - converting to uppercase, AllCapsDemo application, 441
 - DOS, 627
 - drawing, 370-372
 - antialiasing, 372
 - finding font metrics, 372-375
 - unformatted, 627
 - text areas, 263-265**
 - text editors**
 - choosing, 625-626
 - WordPad, 626
 - text fields, 262-263**
 - event handling
 - action events, 345-346
 - item events, 349-350
 - password fields, obscuring, 263
 - TextFrame class, 373-374**
 - this keyword, 124, 138**
 - Thread class, 207**
 - threaded applications**
 - clock, 209
 - writing, 207-209
 - threads**
 - animation, controlling, 207
 - creating, 208-209
 - run() methods, 209
 - loops, 214
 - running, 209
 - terminating, 214
 - threads command (jdb), 654**
 - Throwable class, 190**
 - throwing exceptions, 190, 198-199, 203**
 - explicit, 200
 - implicit, 200
 - inheritance issues, 202
 - throws keyword, 199-200
 - throws keyword, 199-201**
 - timeout values, sockets, 473**
 - TimeServer application**
 - designing, 477-479
 - testing, 479-480
 - TimeServer class, 478-479**
 - Tolksdorf, Robert, 638**
 - ToolBar.java application, 290**
 - toolbars, 289**
 - components, adding, 290
 - dockable toolbars, 290
 - orientation, 289
 - tools**
 - development, selecting, 614
 - functions, modifying, 636

top-level classes, 179

- defining, 181

toUpperCase() method, 74**transient variables, 455-456****transport-layer socket implementation, 477****trimToSize() method, 231****troubleshooting, 654. See also debugging, 654**

- arrays, 95
- compiling, 630
- errors, fatal, 192
- for loops, 109
- methods, overloaded, 133
- scope, variables, 125
- SDK 1.4

- configuring, 616

- installing, 615

- variables, class, 73

true value (Boolean), 48**try and catch blocks, 192****try...catch blocks**

- example, 193-194
- finally clause, 195-196

try...finally blocks, 196**type inference, 244****types**

- primitive. *See* primitive types
- variables, 41

U**unformatted files, saving, 626****unformatted text, 627****Unicode, 420****Unicode character set, 41**

- escape codes, 49

Unicode Consortium Web site, 50**uniform resource identifier (URI), 529****uniform resource locator (URL), 469****up command (jdb), 654****updating progress bars, 293****uppercase, converting text to (AllCapsDemo application), 441****URI (uniform resource identifier), 529****URL (uniform resource locator), 469****URL objects, 468**

- creating, 469

URL() method, 469**US-ASCII character encoding, 484****user coordinate space, 378****user interfaces, frames, 255****UTF-16 character encoding, 484****UTF-16BE character encoding, 484****UTF-16LE character encoding, 484****UTF-8 character encoding, 484****utilities**

- applet viewer, 640
 - AppInfo sample application, 642-644
 - Applet menu commands, 641-642
- command line, 636
 - options, 636-637
- jar, 648-650
- java interpreter, 637-639
- javac compiler, 639-640
- javadoc, 644
 - AppInfo2 sample application, 645-647
 - tags, 645, 648
- jdb, commands, 653-654
- jdb debugger, 650-651
 - applet debugging, 653
 - application debugging, 651-653
 - breakpoints, 651-653
 - commands, 652-654
 - single-step execution, 651

V**validation checks on input streams, 456-457****valueOf() method, 77****values**

- Boolean, 42, 80
- class variables, modifying, 72
- instance variables, modifying, 70-72
- shared, defining, 43
- to variables, assigning, 43

variables, 38

- access control, 151, 155
 - default access, 151
 - private access, 152-153
 - protected access, 154-155
 - public access, 153-154
- array variables, declaring, 92-93
- assigning values to, 40
- casting, 80-81
 - definition of, 80
- class, 18, 39, 70
 - accessing, 157
 - accessing values, 72
 - defining, 40, 72
 - versus instance, 72
 - modifying values, 72
 - troubleshooting, 73
- class types, 43
- class variables, 121
- CLASSPATH
 - Windows 2000/XP, 632-633
 - Windows 9x/Me, 630-631
- constant variables, 43
 - declaring, 44
- creating, 39
- data types, 42
 - converting to objects, 83
- declaring, 39

- decrementing, 54-56
 - duplicating, 126
 - encapsulation, 151
 - environment, 655
 - final, 159
 - final variables, 159
 - in interfaces, 171
 - incrementing, 54-56
 - instance, 18, 39, 70
 - accessing values, 70
 - length, 95
 - modifying, 71
 - modifying values, 70-72
 - testing, 71
 - instance variables, defining, 120-121
 - interface type, 169
 - local, 39
 - naming, 40-41
 - overflow, 62
 - PATH
 - Windows 2000/XP, 623-625
 - Windows 9x/Me, 622-623
 - private variables, object serialization, 464
 - protecting, 162
 - scope, 98, 125-126
 - lexical scope, 115
 - troubleshooting, 125
 - static, 157-159
 - transient variables, 455-456
 - types, 41
 - to values, assigning, 43
 - Vector class, 231**
 - vectors, 231**
 - verbose option (javac), 640**
 - @version tag (javadoc), 645**
 - viewing documents, HTML, 641**
 - virtual machine (java), 637-639**
 - visible frames, 252**
 - void return type (methods), 123**
 - VolcanoRobot class example, attributes, 21**
- ## W
- web servers, Java Web Start (supporting), 405-406**
 - WebReader class, 469-470**
 - website**
 - for this book, 603
 - Java, 636
 - Java 2 documentation, 46
 - Java Plug-in, 641
 - Naviseek, 24
 - Pardon My Icons, 261
 - RSS Advisory Board, 524
 - Unicode Consortium, 50
 - xmethods.com, 547
 - well-formed XML data, 524**
 - well-known ports, 477**
 - while loops, 111-113**
 - overview, 111
 - windowActivated() method, 357**
 - windowClosed() method, 357**
 - windowClosing() method, 357**
 - windowDeactivated() method, 357**
 - windowDeiconified() method, 357**
 - windowIconified() method, 357**
 - WindowListener, 357**
 - WindowListener event listener, 340**
 - windowOpened() method, 357**
 - windows**
 - event handling, 357
 - frames
 - displaying, 252
 - exiting, 253
 - locations, 252
 - sizing, 251
 - user interfaces, 255
 - visible, 252
 - GUI, resizing, 306
 - Java programs
 - compiling in, 628-630
 - running in, 628-630
 - MS-DOS Prompt, 618
 - Swing, resizing, 306
 - Windows, SDK 1.4 (installing on), 615-617**
 - Windows 2000/XP**
 - CLASSPATH variable, 632-633
 - PATH variable, 623-625
 - Windows 9x/Me**
 - CLASSPATH variable, 630-631
 - PATH variable, 622-623
 - wizards, SDK Setup, 615**
 - word processors, 625-626**
 - WordPad, 626**
 - wrap() method, 481**
 - wrapper classes, 128**
 - write() method, 421, 428, 439, 450**
 - writeBoolean() method, 450**
 - writeByte() method, 450**
 - writeBytes() method, 450**
 - WriteBytes.java application, 425-426**
 - writeChar() method, 450**
 - writeChars() method, 450**
 - writeDouble() method, 450**
 - writeFloat() method, 450**
 - writeln() method, 435, 450**
 - writeLong() method, 450**
 - writeObject() method, 450**
 - WritePrimes.java application, 434**
 - Writer class, 463**
 - writeShort() method, 450**
 - writeStream() method, 430**

writing

- apps
 - Android, 567-572
 - threaded, 207-209
- bytes, multiple, 425
- to sockets, 473
- to streams, 421
 - buffered streams, 428, 430
 - character streams, 439-440
 - file output streams, 425-426
 - output streams, 450
- text files, 439-440

X-Y-Z**xmethods.com, 547****XML**

- dialects, designing, 525-526
- documents, formatting, 536-538
- DomainEditor application, creating, 532, 535
- DTD, 525
- elements, 523
- namespaces, 529
- processing
 - with Java, 526
 - with XOM, 526-528
- RSS feed application, creating, 528-531
- tags, 523
- well-formed data, 524

XML-RPC, 546-547

- implementation, selecting, 550-551
- requests, responding to, 549-550
- requests, sending, 548-549
- SOAP, 547
- web service
 - creating, 555-560
 - selecting, 552-555

XOM

- evaluating, 538-540
- XML, processing, 526-528

XOR operator, 57**XYZ color system, 375**