Sams Teach Yourself

Java™

in 24 Hours

Seventh Edition
# Table of Contents

## Introduction

<table>
<thead>
<tr>
<th>PART I: Getting Started</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOUR 1: Becoming a Programmer</td>
<td>4</td>
</tr>
<tr>
<td>Choosing a Language</td>
<td>6</td>
</tr>
<tr>
<td>Telling the Computer What to Do</td>
<td>8</td>
</tr>
<tr>
<td>How Programs Work</td>
<td>9</td>
</tr>
<tr>
<td>When Programs Don’t Work</td>
<td>10</td>
</tr>
<tr>
<td>Choosing a Java Programming Tool</td>
<td>10</td>
</tr>
<tr>
<td>Installing a Java Development Tool</td>
<td>10</td>
</tr>
</tbody>
</table>

## HOUR 2: Writing Your First Program

| What You Need to Write Programs | 15 |
| Creating the Saluton Program | 16 |
| Beginning the Program | 16 |
| Storing Information in a Variable | 20 |
| Saving the Finished Product | 21 |
| Compiling the Program into a Class File | 22 |
| Fixing Errors | 23 |
| Running a Java Program | 24 |

## HOUR 3: Vacationing in Java

| First Stop: Oracle | 29 |
| Going to School with Java | 32 |
| Lunch in JavaWorld | 34 |
| Watching the Skies at NASA | 36 |
| Getting Down to Business | 37 |
| Stopping by SourceForge for Directions | 39 |
| Running Java on Your Phone | 41 |

## HOUR 4: Understanding How Java Programs Work

| Creating an Application | 47 |
| Sending Arguments to Applications | 49 |
| The Java Class Library | 51 |

## PART II: Learning the Basics of Programming

| HOUR 5: Storing and Changing Information in a Program | 59 |
| Statements and Expressions | 60 |
| Assigning Variable Types | 64 |
| Naming Your Variables | 65 |
| Storing Information in Variables | 66 |
| All About Operators | 70 |
| Using Expressions | 77 |

## HOUR 6: Using Strings to Communicate

| Storing Text in Strings | 77 |
| Displaying Strings in Programs | 78 |
| Using Special Characters in Strings | 79 |
| Pasting Strings Together | 80 |
| Using Other Variables with Strings | 81 |
| Advanced String Handling | 82 |
| Presenting Credits | 84 |

## HOUR 7: Using Conditional Tests to Make Decisions

| if Statements | 91 |
| if-else Statements | 95 |
| switch Statements | 96 |
| The Ternary Operator | 98 |
| Watching the Clock | 99 |

## HOUR 8: Repeating an Action with Loops

| for Loops | 107 |
| while Loops | 110 |
| do-while Loops | 111 |
| Exiting a Loop | 112 |
| Naming a Loop | 113 |
| Testing Your Computer Speed | 115 |

## PART III: Working with Information in New Ways

| HOUR 9: Storing Information with Arrays | 122 |
| Creating Arrays | 123 |
| Using Arrays | 125 |
| Multidimensional Arrays | 126 |
| Sorting an Array | 128 |
| Counting Characters in Strings | 135 |

## HOUR 10: Creating Your First Object

| How Object-Oriented Programming Works | 136 |
| Objects in Action | 138 |
| What Objects Are | 139 |
| Understanding Inheritance | 140 |
| Building an Inheritance Hierarchy | 141 |
| Converting Objects and Simple Variables | 146 |
| Creating an Object | 146 |

## HOUR 11: Describing What Your Object Is Like

| Creating Variables | 153 |
| Creating Class Variables | 156 |
| Creating Behavior with Methods | 157 |
| Putting One Class Inside Another | 162 |
About the Author

Rogers Cadenhead is a writer, computer programmer, and web developer who has written more than 20 books on Internet-related topics, including Sams Teach Yourself Java in 21 Days. He maintains the Drudge Retort and other websites that receive more than 20 million visits a year. This book’s official website is at www.java24hours.com.

Dedication

I began programming as a 13-year-old on a Timex Sinclair 1000, a computer with a 3.25 MHz processor and 2KB of memory that used a TV as a monitor. I’d like to dedicate this book to the person who bought that computer and never complained when I immediately stole it from him—my dad, Roger Cadenhead, Sr. Thanks, Dad! That led me to this.

Acknowledgments

To the folks at Sams and Pearson—especially Mark Taber, Tonya Simpson, Seth Kerney, Barbara Hacha, and Boris Minkin. No author can produce a book like this on his own. Their excellent work will give me plenty to take credit for later.

To my wife, Mary, and my sons, Max, Eli, and Sam.
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.

E-mail: feedback@samspublishing.com

Mail: Reader Feedback
Sams Publishing
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

As the author of computer books, I spend a lot of time lurking in the computer section of bookstores, observing the behavior of readers while I’m pretending to read the latest issue of *In Touch Weekly* magazine.

Because of my research, I’ve learned that if you have picked up this book and turned to the introduction, I only have 13 more seconds before you put it down and head to the coffee bar for a double-tall-decaf-skim-with-two-shots-of-vanilla-hold-the-whip latte.

So I’ll keep this brief: Computer programming with Java is easier than it looks. I’m not supposed to tell you that because thousands of programmers have used their Java skills to get high-paying jobs in software development, web application programming, and mobile app creation. The last thing any programmer wants is for the boss to know that anyone with persistence and a little free time can learn this language, the most popular programming language on the planet. By working your way through each of the one-hour tutorials in *Sams Teach Yourself Java in 24 Hours*, you’ll be able to learn Java programming quickly.

Anyone can learn how to write computer programs—even if you can’t program a DVR. Java is one of the best programming languages to learn because it’s a useful, powerful, modern technology that’s embraced by programmers around the world.

This book is aimed at nonprogrammers, new programmers who hated learning the subject, and experienced programmers who want to get up to speed swiftly with Java. It uses Java 8, the brand-new version of the language.

Java is an enormously popular programming language because of the things it makes possible. You can create programs that feature a graphical user interface, design software that makes the most of the Internet, connect to web services, create an app that runs on an Android phone or tablet, and more.

This book teaches Java programming from the ground up. It introduces the concepts in English instead of jargon with step-by-step examples of working programs you will create. Spend 24 hours with this book and you’ll be writing your own Java programs, confident in your ability to use the language and learn more about it. You also will have skills that are becoming increasingly important—such as network computing, graphical user interface design, and object-oriented programming.
These terms might not mean much to you now. In fact, they’re probably the kind of thing that makes programming seem intimidating and difficult. However, if you can use a computer to create a photo album on Facebook, pay your taxes, or work an Excel spreadsheet, you can learn to write computer programs by reading *Sams Teach Yourself Java in 24 Hours*.

**NOTE**

At this point, if you would rather have coffee than Java, please reshelve this book with the front cover facing outward on an endcap near a lot of the store’s foot traffic.
As you learned during Hour 1, “Becoming a Programmer,” a computer program is a set of instructions that tell a computer what to do. These instructions are given to a computer using a programming language.

During this hour, you create your first Java program by entering it into a text editor. When that’s done, you save the program, compile it, and test it out. Then you break it on purpose and fix it again, just to show off.

What You Need to Write Programs

As explained in Hour 1, to create Java programs, you must have a programming tool that supports the Java Development Kit (JDK) such as the NetBeans integrated development environment (IDE). You need a tool that can compile and run Java programs and a text editor to write those programs.

With most programming languages, computer programs are written by entering text into a text editor (also called a source code editor). Some programming languages come with their own editor. NetBeans includes its own editor for writing Java programs.

Java programs are simple text files without any special formatting, such as centered text or boldface text. The NetBeans source code editor functions like a simple text editor with some extremely useful enhancements for programmers. Text turns different colors as you type to identify different elements of the language. NetBeans also indents lines properly and provides helpful programming documentation inside the editor.

THIS HOUR’S TO-DO LIST:

- Type a Java program into a text editor.
- Organize a program with bracket marks.
- Store information in a variable.
- Display the information stored in a variable.
- Save, compile, and run a program.
Because Java programs are text files, you can open and edit them with any text editor. You could write a Java program with NetBeans, open it in Windows Notepad and make changes, and open it again later in NetBeans without any problems.

Creating the Saluton Program

The first Java program that you create will display a traditional greeting from the world of computer science: “Saluton mondo!”

To prepare for the first programming project in NetBeans, if you haven’t already done so, create a new project called Java24 by following these steps:

2. Choose the project category Java and the project type Java Application and then click Next.
3. Enter Java24 as the project’s name. (If you created a project with this name previously, you see the error message “Project folder already exists and is not empty.”)
4. Deselect the Create Main Class check box.
5. Click Finish.

The Java24 project is created in its own folder. You can use this project for the Java programs you write as you progress through this book.

Beginning the Program

NetBeans groups related programs together into a project. If you don’t have the Java24 project open, here’s how to retrieve it:

2. Find and select the NetBeansProjects folder (if necessary).
3. Choose Java24 and click Open Project.

The Java24 project appears in the Projects pane next to a coffee cup icon and a + sign that can be expanded to see the files and folders that the project contains.
To add a new Java program to the currently open project, choose File, New File. The New File Wizard opens, as shown in Figure 2.1.

The Categories pane lists the different kinds of Java programs you can create. Click the Java folder in this pane to see the file types that belong to this category. For this first project, choose the Empty Java File type and click Next.

A New Empty Java File dialog opens. Follow these steps to begin writing the program:

1. In the Class Name field, enter `Saluton`.
2. In the Package field, enter `com.java24hours`.
3. Click Finish.

So you can begin working right away on your program, an empty file named `Saluton.java` opens in the source code editor. Using the editor, begin your Java programming career by entering each line from Listing 2.1. These statements are called the program’s source code.

**CAUTION**

Don’t enter the line number and colon at the beginning of each line—these are used in this book to reference specific line numbers.
LISTING 2.1  The Saluton Program

```
1: package com.java24hours;
2: 
3: class Saluton {
4:     public static void main(String[] arguments) {
5:         // My first Java program goes here
6:     }
7: }
```

Make sure to capitalize everything exactly as shown, and use your spacebar or Tab key to insert the blank spaces in front of Lines 4–6. When you’re done, choose File, Save to save the file.

At this point, Saluton.java contains the bare-bones form of a Java program. You will create many programs that start exactly like this one, except for the word Saluton on Line 3. This word represents the name of your program and changes with each program you write. Line 5 should make sense to you, because it’s a sentence in actual English. The rest is probably new to you.

**The class Statement**

The first line of the program is the following:

```
package com.java24hours;
```

A package is a way to group Java programs together. This line tells the computer to make com.java24hours the package name of the program.

After a blank line, the third line is this:

```
class Saluton {
```

Translated into English, it means, “Computer, give my Java program the name Saluton.”

As you might recall from Hour 1, each instruction you give a computer is called a statement. The class statement is the way you give your computer program a name. It’s also used to determine other things about the program, as you will see later. The significance of the term class is that Java programs also are called classes.

In this example, the program name Saluton matches the document’s filename, Saluton.java. A Java program must have a name that matches the first part of its filename and should be capitalized the same way.
If the program name doesn’t match the filename, you get an error when you try to compile some Java programs, depending on how the class statement is being used to configure the program.

**What the main Statement Does**

The next line of the program is the following:

```java
public static void main(String[] arguments) {
```

This line tells the computer, “The main part of the program begins here.” Java programs are organized into different sections, so there needs to be a way to identify the part of a program that is executed first when the program is run.

The main statement is the entry point to most Java programs. The exceptions are applets, programs that are run on a web page by a web browser; servlets, programs run by a web server; and apps, programs run by a mobile device.

Most programs you write during upcoming hours use main as their starting point. That’s because you run them directly on your computer. Applets, apps, and servlets are run indirectly by another program or device.

To differentiate them from these other types, the programs that you run directly are called applications.

**Those Squiggly Bracket Marks**

In the Saluton program, Lines 3, 4, 6, and 7 contain a squiggly bracket mark of some kind—either a { or a }. These brackets are a way to group lines of your program (in the same way that parentheses are used in a sentence to group words). Everything between the opening bracket { and the closing bracket } is part of the same group.

These groupings are called blocks. In Listing 2.1, the opening bracket on Line 3 is associated with the closing bracket on Line 7, which makes your entire program a block. You use brackets in this way to show the beginning and end of a program.

Blocks can be located inside other blocks (just as parentheses are used in this sentence (and a second set is used here)). The Saluton program has brackets on Line 4 and Line 6 that establish another block.
This block begins with the main statement. The lines inside the main statement's block will be run when the program begins.

The following statement is the only thing located inside the block:

```java
// My first Java program goes here
```

This line is a placeholder. The // at the beginning of the line tells the computer to ignore this line because it was put in the program solely for the benefit of humans who are looking at the source code. Lines that serve this purpose are called comments.

Right now, you have written a complete Java program. It can be compiled, but if you run it, nothing happens. The reason is that you haven’t told the computer to do anything yet. The main statement block contains only a single comment, which is ignored by the computer. You must add some statements inside the opening and closing brackets of the main block.

## Storing Information in a Variable

In the programs you write, you need a place to store information for a brief period of time. You can do this by using a variable, a storage place that can hold information such as integers, floating-point numbers, true-false values, characters, and lines of text. The information stored in a variable can change, which is how it gets the name variable.

In the `Saluton.java` file, replace Line 5 with the following:

```java
String greeting = "Saluton mondo!";
```

This statement tells the computer to store the text “Saluton mondo!” in a variable called `greeting`.

In a Java program, you must tell the computer what type of information a variable will hold. In this program, `greeting` is a string—a line of text that can include letters, numbers, punctuation, and other characters. Putting `String` in the statement sets up the variable to hold string values.

When you enter this statement into the program, a semicolon must be included at the end of the line. Semicolons end each statement in a Java program. They’re like the period at the end of a sentence. The computer uses them to determine when one statement ends and the next one begins.
Putting only one statement on each line makes a program more understandable (for us humans).

**Displaying the Contents of a Variable**

If you run the program at this point, it still seems like nothing happens. The command to store text in the `greeting` variable occurs behind the scenes. To make the computer show that it is doing something, you can display the contents of that variable.

Insert another blank line in the `Saluton` program after the `String greeting = "Saluton mondo!"` statement. Use that empty space to enter the following statement:

```java
System.out.println(greeting);
```

This statement tells the computer to display the value stored in the `greeting` variable. The `System.out.println` statement makes the computer display information on the system output device—your monitor.

Now you’re getting somewhere.

**Saving the Finished Product**

Your program should now resemble Listing 2.2, although you might have used slightly different spacing in Lines 5–6. Make any corrections that are needed and save the file (by choosing File, Save).

**LISTING 2.2  The Finished Version of the Saluton Program**

```java
1: package com.java24hours;
2: 
3: class Saluton {
4:     public static void main(String[] arguments) {
5:         String greeting = "Saluton mondo!";
6:         System.out.println(greeting);
7:     }
8: }
```

When the computer runs this program, it runs each of the statements in the `main` statement block on Lines 5 and 6. Listing 2.3 shows what the program would look like if it was written in the English language instead of Java.
LISTING 2.3  A Line-by-Line Breakdown of the Saluton Program

1: Put this program in the com.java24hours package.
2: 
3: The Saluton program begins here:
4: The main part of the program begins here:
5: Store the text "Saluton mondo!" in a String variable named greeting
6: Display the contents of the variable greeting
7: The main part of the program ends here.
8: The Saluton program ends here.

Listing 2.4 shows what the program would look like if written in Klingon, the language of the warrior race from Star Trek.

LISTING 2.4  The Saluton Program in Klingon

1: This program belongs to the house of com.java2hours!
2: 
3: Begin the Saluton program here if you know what's good for you!
4: The main part of the program begins here with honor!
5: Store the gibberish "Saluton mondo!" in a String variable called greeting!
6: Display this gibberish from a tongue inferior to Klingon!
7: End the main part of the program here to avoid my wrath!
8: End the Saluton program now and be grateful you were spared!

Compiling the Program into a Class File

Before you can run a Java program, you must compile it. When you compile a program, the instructions given to the computer in the program are converted into a form the computer can better understand.

NetBeans compiles programs automatically as they are saved. If you typed everything as shown in Listing 2.2, the program compiles successfully.

A compiled version of the program, a new file called Saluton.class, is created. All Java programs are compiled into class files, which are given the .class file extension. A Java program can be made up of several classes that work together, but in a simple program such as Saluton only one class is needed.

The compiler turns Java source code into bytecode, a form that can be run by the Java Virtual Machine (JVM).
Fixing Errors

As you compose a program in the NetBeans source editor, errors are flagged with a red alert icon to the left of the editor pane, as shown in Figure 2.2.

![Error Icon](image)

FIGURE 2.2
Spotting errors in the source editor.

The icon appears on the line that triggered the error. You can click this icon to display an error message that explains the compiler error with these details:

- The name of the Java program
- The type of error
- The line where the error was found

Here's an example of an error message you might see when compiling the Saluton program:

cannot find symbol.
symbol : variable greeting
location: class Saluton

The error is the first line of the message: “cannot find symbol.” These messages often can be confusing to new programmers. When the error message doesn’t make sense to you, don’t spend much time trying to figure it out. Instead, take a look at the line where the error occurred and look for the most obvious causes.
For instance, can you determine what’s wrong with the following statement?

```java
System.out.println(greting);
```

The error is a typo in the variable name, which should be `greeting` instead of `greting` (Add this typo on purpose in NetBeans to see what happens.)

If you get error messages when creating the `Saluton` program, double-check that your program matches Listing 2.2 and correct any differences you find. Make sure that everything is capitalized correctly and all punctuation marks such as {, }, and ; are included.

Often, a close look at the line identified by the error message is enough to reveal the error (or errors) that need to be fixed.

**Running a Java Program**

To see whether the `Saluton` program does what you want, run the class with the Java Virtual Machine, the interpreter that runs all Java code. In NetBeans, choose the menu command Run, Run File. An Output pane opens below the source code editor. In this pane, if there are no errors, the program displays the output, as shown in Figure 2.3.
If you see the text “Saluton Mondo!” you have just written your first working Java program! Your computer has just greeted the world—a tradition in the computer programming field that’s as important to many of us as caffeine, short-sleeved dress shirts, and *Call of Duty*.

You might be asking yourself why “Saluton mondo!” is a traditional greeting. The phrase means “Hello world!” in Esperanto, an artificial language created by Ludwig Zamenhof in 1887 to facilitate international communication. It’s only a traditional greeting in the sense that I’m trying to start that tradition.

**Summary**

During this hour, you got your first chance to create a Java program. You learned that to develop a Java program you need to complete these four basic steps:

1. Write the program with a text editor or a tool such as NetBeans.
2. Compile the program into a class file.
3. Tell the Java Virtual Machine to run the class.
4. Call your mother.

Along the way, you were introduced to some basic computer programming concepts such as compilers, interpreters, blocks, statements, and variables. These will become clearer to you in successive hours. As long as you got the *Saluton* program to work during this hour, you’re ready to proceed.

(The fourth step has nothing to do with Java programming. It’s just something my mother suggested I put in the book.)
Workshop

Q&A

Q. How important is it to put the right number of blank spaces on a line in a Java program?

A. It’s completely unimportant as far as the computer is concerned. Spacing is strictly for the benefit of people looking at a computer program—the Java compiler couldn’t care less. You could have written the Saluton program without using blank spaces or used the Tab key to indent lines, and it would compile successfully.

Although the number of spaces in front of lines isn’t important, you should use consistent spacing and indentation in your Java programs. Why? Because spacing makes it easier for you to see how a program is organized and to which programming block a statement belongs.

The programs you write must be understandable to other programmers, including yourself when you look at the code weeks or months later to fix a bug or make an enhancement. Consistency in spacing and indentation are part of what’s called a programming style. Good programmers adopt a style and practice it in all their work.

Q. A Java program has been described as a class and as a group of classes. Which is it?

A. Both. The simple Java programs you create during the next few hours are compiled into a single file with the extension .class. You can run these with the Java Virtual Machine. Java programs also can be made up of a set of classes that work together. This topic is fully explored during Hour 10, “Creating Your First Object.”

Q. If semicolons are needed at the end of each statement, why does the comment line // My first Java program goes here not end with a semicolon?

A. Comments are completely ignored by the compiler. If you put // on a line in your program, this tells the Java compiler to ignore everything to the right of the // on that line. The following example shows a comment on the same line as a statement:

```
System.out.println(greeting);  // hello, world!
```

Q. I couldn’t find any errors in the line where the compiler noted an error. What can I do?
A. The line number displayed with the error message isn’t always the place where an error needs to be fixed. Examine the statements that are directly above the error message to see whether you can spot any typos or other bugs. The error usually is within the same programming block.

Q. How can I visit Antarctica?

A. If you’re not willing to become a scientific researcher or a support staffer such as a cook, electrician, or doctor, you can become one of the 10,000 people who visit the frozen continent annually as tourists. Flyovers are available from Australia, New Zealand, and South America and cost around $1,000 per person.

Several cruise ships visit for a trip lasting from 10 days to three weeks, the most expensive of which is around $25,000. Some cruises offer a chance to kayak or hike among penguins, visit icebergs, and even camp overnight.

The Polar Cruises website at www.polarcruises.com provides more information for prospective Antarctica visitors.

The British Antarctic Survey offers a piece of advice for visitors: “Do not walk onto glaciers or large snowfields unless properly trained.”

Quiz

Test your knowledge of the material covered in this hour by answering the following questions.

1. When you compile a Java program, what are you doing?
   A. Saving it to a disk
   B. Converting it into a form the computer can better understand
   C. Adding it to your program collection

2. What is a variable?
   A. Something that wobbles but doesn’t fall down
   B. Text in a program that the compiler ignores
   C. A place to store information in a program
3. What is the process of fixing errors called?
   
   A. Defrosting
   
   B. Debugging
   
   C. Decomposing

**Answers**

1. B. Compiling a program converts a .java file into a .class file or a set of .class files.

2. C. Variables are one place to store information; later you learn about others such as arrays and constants. Weebles wobble but they don’t fall down, and comments are text in a program that the compiler ignores.

3. B. Because errors in a computer program are called bugs, fixing those errors is called debugging. Some programming tools come with a tool called a debugger that helps you fix errors. NetBeans has one of debest debuggers.

**Activities**

If you’d like to explore the topics covered in this hour a little more fully, try the following activities:

- You can translate the English phrase “Hello world!” into other languages using the Google Translator at http://translate.google.com. Write a program that enables your computer to greet the world in a language such as French, Italian, or Portuguese.

- Go back to the Saluton program and add one or two errors. For example, take a semicolon off the end of a line or change the text println on one line to print1n (with a number 1 instead of the letter L). Save the program and try to compile it; then compare the error messages you see to the errors you caused.

To see solutions to these activities, visit the book’s website at www.java24hours.com.
INDEX

NUMERICS
2D graphics, 362
arcs, 364-365, 373
circles, 364
ellipses, 364
lines, 362
PiePanel application, 365-366
PiePanel.java source code, 370
PieSlice class, 367-368
rectangles, 363

Symbols
$ (dollar sign), 64
\ (backslash), 79
{} (braces), 19-20, 59, 94, 104
[] (brackets), 122
- (decrement operator), 67
/ (division operator), 66
* (double quotation mark), 61
// (double slashes), 20
== (equality operator), 93
= (equal sign), 62, 65
/ (forward slash) character, 330
> (greater than operator), 93
!= (inequality operator), 93
< (less than operator), 92-93
- (minus sign), 66-67
* (multiplication operator), 67
n (newline character), 200
% operator, 67
+= operator, 81
| (pipe) characters, 282
+ (plus sign), 66, 80-81
? (question mark), 98
; (semicolon), 20, 26, 114
' (single quotation marks), 61, 79
[] (square brackets), 122
// (two slash characters), 287
_ (underscore), 64

A
Absolute program, 40
Abstract Windowing Toolkit. See AWT
access control
definition of, 154
public methods, 158
variables, 155
ActionListener interface, 228, 302
actionPerformed() method, 229, 239, 305-306
add() method, 175, 203
addActionListener() method, 228
addChangeListener() method, 249
add(Component) method, 253
adding
emulators, 382
objects, 264
addItemListener() method, 230-232
addition operator (+), 66
addKeyListener() method, 230
addOneToField() method, 238
addSlice() method, 368
Agile Java Development with Spring, Hibernate and Eclipse, 414
Android
applications
configuring AVDs, 382
creating, 377-381
Debug Configurations, 384
debugging, 399
design, 388-391
interface design, 392-396
manifest files, 391-392
navigating, 379-380
overview of, 375-377
running, 385-387
writing Java code, 396-401
IDEs, 421
phones
configuring, 424-426
running Java on, 41
plug-ins
Eclipse, 376
installing, 423
programming, 421-422
resources, 391
Android Programming Unleashed, 414
Android Virtual Devices. See AVDs
AndroidManifest.xml file, 380, 390
Angry Birds application, 41
annotations, 346-348
anonymous inner classes, 316-320
Apache Project, 52
apostrophes (‘), 61
app_name string resource, 381
applets, 29
definition of, 29, 47
event handling, 227
actionPerformed() method, 229
check boxes, 230
combo boxes, 230
event listeners, 227-228
keyboard events, 232
real-word examples, 32
Revolve, 301
saving, 8
security, digital signatures, 35
threaded, 301

applications

Android
configuring AVDs, 382
creating, 377-381
Debug Configurations, 384
debugging, 399
design, 388-391
interface design, 392-396
manifest files, 391-392
navigating, 379-380
overview of, 375-377
running, 385-387
writing code, 396-401

Angry Birds, 41
arguments, 49-51, 57
autodialers, 137
BlankFiller.java source code, 50
block statements, 93-95
Clock, 101
colors, 359
  RGB values, 361
  setting, 361
compiling, 22-23, 49
Configurator.java, 342
Console, 336
constructors, 302-303
creating, 47-48
Crisis, 213
definition of, 47
deploying, 424
design, 5
executing, 8-9
Fonts, 359
formatting, 217, 220-222
Game source code, 94
graphics, 233-242
ID3Reader, 332
KeyView.java, 232
LeaderActivity, 396-400
Lotto Madness, 234
  aplet version, 242
event listeners, 234
LottoEvent.java class, 235, 238
methods, 238-239
source code listing, 240-242

multithreading, 36
NetBeans
  running, 410-411
troubleshooting, 411-412
NumberDivider, 282-284
PageCatalog, 288-291
PieFrame, 371-372
PiePanel, 365-370
PlanetWeight, 71
ReadConsole, 335
Root
  compiling, 49
  source code, 48
running, 24-25
Saluton, 16-18
classes, 18-19
compiling, 22-23
greeting variable, 21
line-by-line breakdown, 21
main() block, 19
saving, 21
writing, 16-25
saving, 8. See also applets
SquareRootClient, 353-355
SquareRootServer, 348
starting, 16-18
threading, 295-299
Tool, 254-256
troubleshooting, 9, 23
variables
  char variables, 61-62
  floating-point variables, 61
  int statement, 60
  integer variables, 61
  string variables, 61-62

Virus, 165
class constructor, 160
getSeconds() method, 158
setSeconds() method, 158
showVirusCount(), 161
tauntUser() method, 159
VirusLook source code, 166
Wheel of Fortune, 129
writing, 15-16

applying
annotations, 346-348
arrays, 123-125
Color class, 360
components, 190
expressions, 70-72
Font class, 359-360
inner classes, 313-320
methods, 165-167
NetBeans, 405
  creating new projects, 406-408
  formatting classes, 408-410
  installing, 405
  running, 410-411
  troubleshooting, 411-412
objects, 177-178
Package Explorer, 380
threads, 301-303
variables, 165-167

Arc2D class, 364-365
arcs, drawing, 364-365, 373
arguments, 57
  applications, 49-51
  methods, 159
ArrayIndexOutOfBoundsException
  Exception class, 278
arrayoutofbounds errors, 124

arrays
  applying
    annotations, 346-348
    arrays, 123-125
    Color class, 360
    components, 190
    expressions, 70-72
    Font class, 359-360
    inner classes, 313-320
    methods, 165-167
    NetBeans, 405
      creating new projects, 406-408
      formatting classes, 408-410
      installing, 405
      running, 410-411
      troubleshooting, 411-412
    objects, 177-178
    Package Explorer, 380
    threads, 301-303
    variables, 165-167

ASCII character sets, 131
/assets, 380
assigning variables, 60, 65-66
asterisk (*), 67
attributes, 136-141, 153
autoboxing, 146-146
autodialers, 137
AVDs (Android Virtual Devices), 382
AWT (Abstract Windowing Toolkit), 189, 216-217

B
backslash (\), 79
backspaces, 79
Clocks

inner, 162-164
anonymous, 316-320
applying, 313-316
Insets, 216-217
JApplet, 173-175
Java libraries, 51-54
JButton, 195-197
JCheckBox, 198-199
JComboBox, 199-200
JFrame, 191
JLabel, 197-198
JPanel, 203
JScrollPane, 201
JSlider, 247
JTextArea, 200
JTextField, 197-198
Line2D, 362
LottoEvent, 235, 238
methods
applying, 165-167
declaring, 161
testing, 184
private, 150
R, 397
Random, 55
ReadConsole, 335
Rectangle2D, 363
statements, 18-19, 138
subclasses, 140, 148, 175-184
superclasses, 140
testing, 184
Thread, 295
threaded, 296-300
variables, 156
Virus, 153
clearAllFields() method, 238
clients, creating web services, 353-355
Clock application source code, 101
Clocks, 99-102

BASIC (Beginner's All Symbolic Instruction Code), 4, 12
behavior. See also methods
inheritance, 139-141
methods, 157
objects, 136
benchmarks, 115-117
blank spaces in source code, 26
BlankFiller.java application, 50
blocks, 19-20
braces ({}), 19-20, 59, 94, 104
books, Java-related, 413
Boole, George, 63
Boolean variables, 63-64
BorderLayout manager, 214-216
borders, Insets class, 216-217
BoxLayout manager, 215
braces ({}), 19-20, 59, 94, 104
brackets [ ], 122
break statement, 97, 104, 112
breaking loops, 112
browsers, Java Plug-in, 33
buffered input streams, 334-339
bugs, 9. See also debugging
buttons, creating, 195-196
bytecode, 330
bytes, 330, 343

C
C#, 4
C++, 4-5, 12
Cadenhead, Rogers, 413
calculating percentages, 206
calling web services, 355
cannot resolve symbol (error message), 23
career opportunities, 417
carriage returns, 79
case
modifying, 83, 87
statements, 96
variable names, 64
casting, 141
definition of, 141
destinations, 141
objects, 143
sources, 141
variables, 142
catch statement, 303, 310
catching
errors, 303
exceptions, 277-278
multiple exceptions, 282-284, 290
PageCatalog sample application, 288-291
try-catch blocks, 279-291
try-catch-finally blocks, 284
change listeners, 249-250
ColorSlide sample application, 252
registering objects as, 249
ChangeListener interface, 249
char variables, declaring, 61-62, 77
circulars, drawing, 364
.class file extension, 26
classes, 136
applets, 175
Arc2D, 364-365
ArrayIndexOutOfBoundsException
Exception, 278
Color, 361
Console, 336
declaring, 301-302
documentation, 415
Ellipse2D, 364
encapsulation, 158
ErrorCorrectionModem, 139
Exception, 278
files, 148, 330-331
FileInputStream, 336
FileOutputStream, 336
Graphics2D, 362
arcs, 364-365, 373
circles, 364
eccentricals, 364
lines, 362
rectangles, 363
hierarchies, 173, 186
inheritance, 139-141, 150, 173-176
inner, 162-164
anonymous, 316-320
applying, 313-316
Insets, 216-217
JApplet, 173-175
Java libraries, 51-54
JButton, 195-197
JCheckBox, 198-199
JComboBox, 199-200
JFrame, 191
JLabel, 197-198
JPanel, 203
JScrollPane, 201
JSlider, 247
JTextArea, 200
JTextField, 197-198
Line2D, 362
LottoEvent, 235, 238
methods
applying, 165-167
declaring, 161
Modem, 138
NetBeans, 408-410
objects
loops, 180-182
storing, 178-180
PieSlice, 367-368
Point, 182
Point3D
code listing, 183
creating, 183-184
testing, 184
private, 150
R, 397
Random, 55
ReadConsole, 335
Rectangle2D, 363
statements, 18-19, 138
subclasses, 140, 148, 175-184
superclasses, 140
testing, 184
Thread, 295
threaded, 296-300
variables, 156
Virus, 153
clearAllFields() method, 238
clients, creating web services, 353-355
Clock application source code, 101
Clocks, 99-102
close() method, 337
closing streams, 337
closures, 321-326
code
Android applications, 396-401
annotations, formatting, 346-348
code listings
Battlepoint.java application, 267
Benchmark.java application, 116
BlankModem.java application, 50
CableMode.java class, 147
Calculator.java application, 279
Catalog.java application, 315
Clock application, 99, 101
ColorFrame.java application, 323
ColorSliders.java application, 250
Commodity program, 97
Configurator.java application, 341
ConfigWriter.java, 338
Console application, 336
Console.java application, 335
Credits application, 85
Crisis application, 213
Crisis.java application, 212
Dice.java application, 54
DslModem class, 148
FontMapper.java application, 272
FreeSpaceFrame.java application, 207
Game program, 94
HomePage.java application, 289
ID3Reader.java application, 333
KeyView.java application, 232
KeyViewer.java application, 231
LeaderActivity.java application, 399
LinkRotator.java application, 306
LottoEvent.java application, 235
LottoEvent.java class, 235, 238
LottoMadness application, 220, 240-242
LottoMadness.java application, 218
Modem.java class, 147
NameSorter.java application, 127
NewCalculator.java application, 281
NewColorFrame.java application, 324
NewKeyViewer.java application, 319
NewRoot.java application, 144
Nines application, 109
NumberDivider.java application, 283
PageCatalog.java application, 290
PieFrame application, 370
PiePanel.java source code, 369-370
PlanetWeight application, 71
Playback.java application, 196
Point3D class, 183
PointTester.java application, 184
PrimeFinder.java application, 298
Root application, 48
Saluton application, 18, 21
SalutonFrame.java application, 193
SpaceRemover.java application, 125
SquareRootClient.java application, 354
SquareRootServer.java application, 347
SquareRootServerImpl.java application, 349
SquareRootServerPublisher.java application, 350
StringLister.java application, 181
Tableframe.java application, 259
TestModems class, 148
Tool.java application, 254
Variable application, 62
Virus.java application, 165
VirusLab.java application, 166
Web Service Description Language Contract, 351
Wheel.java application, 129
Color class, 360-361
colors, 359
Color class, 360
Font class, 359-360
RGB values, 361
setting, 361
ColorSliders application, 252
combo boxes
creating, 199-200
event handling, 230
commands, 6, 49
comments, 20, 26
comparing strings, 82
equal/not equal, 93
less/greater than, 92-93
collecting languages, performance, 12
compilers
definition of, 8
javac, 23-24
compiling, 22
applications, 22, 49
Windows, 23
complex for loops, 114-115
components, 190, 247
arranging, 209
buttons, creating, 195-196
change listeners, 249-250
ColorSliders sample application, 252
registering objects as, 249
check boxes
creating, 198-199
event handling, 230
combo boxes, 199-200, 230
creating, 203-208
disabling, 233
enabling, 233
frames, 190-191
adding components to, 195
creating, 191, 194
sizing, 192
image icons, 252-256
labels, creating, 197-198
panels, 203
scroll panes, 201
adding components to, 202
creating, 201
sliders, 247-248
creating, 248
labels, 248
tables, 256-260
text areas, creating, 200
text fields
creating, 197-198
write protecting, 223
TextFields, 197
toolbars, 252-253
creating, 253-255
dockable toolbars, 253
Tool sample application, 254-256
windows, 190-195
computer speed, testing, 115-117
congatating strings, 80
congatation operator (+), 80-81
Conder, Shane, 422
conditionals, 91

Clock application source code, 101
if, 92-93, 104
blocks, 93-95
equal/not equal comparisons, 93
less/greater than comparisons, 92-93
if-else, 95-96
switch, 96
ternary operator (?), 98
configuration properties, reading/writing, 339-342
Configurator.java application, 342
configuring
AVDs (Android Virtual Devices), 382
Debug Configurations, 384
phones, 424-426
variables, 302
Console application, 336
constants, 66
constructor methods, 159-160, 302-303
containers, 190, 203
continue statement, 113
contracts, WSDL, 351
controlling access, 155. See also access control
converting
objects to variables, 141-144
variables to objects, 144
counter variables
definition of, 108
initializing, 108
counting character strings, 128-130
createNewFile() method, 331
Creative Commons, 334
credits, viewing, 84-86
Crisis application, 213
currentThread() method, 305
customizing properties, 395

D

Darcey, Lauren, 422
data structures
array lists, 263-268
hash maps, 269-272
data types. See also types
Boolean, 63-64
byte, 62
debugging
Android applications, 390, 399
definition of, 9
OOP applications, 137
phones, 425
declaring
arrays, 122-126
classes, 301-302
class statement, 18-19
subclasses, 175-184
methods, 157
classes, 161
constructors, 160
public methods, 158
variables, 60
Boolean, 63-64
char, 61-62
long, 63
short, 62
String, 20
Development Configurations, creating, 384
development settings, 386
development tools, 4
Dice program, 53
Dice.java, 54
digital signatures, 35
disabling components, 233
displaying. See viewing
displaySpeed() method, 138-139
division operator (/), 66
do-while loops, 111-112
dockable toolbars, 253
docking toolbars, 256
documentation, 25, 415
Java Class Library, 52
Swing, 261
dollar sign ($), 64
double quotation mark ("), 61
double slashes (//), 20
draw() method, 362
drawing
arcs, 364-365, 373
circles, 364
ellipses, 364
lines, 361-362
pie graphs, 365-366
PiePanel.java source code, 370
PieSlice class, 367-368
rectangles, 363
shapes, 361
drawRoundRect() method, 363-364
drawString() method, 157
development settings, 386
development tools, 4
Dice program, 53
Dice.java, 54
digital signatures, 35
disabling components, 233
displaying. See viewing
displaySpeed() method, 138-139
division operator (/), 66
do-while loops, 111-112
dockable toolbars, 253
docking toolbars, 256
documentation, 25, 415
Java Class Library, 52
Swing, 261
dollar sign ($), 64
double quotation mark ("), 61
double slashes (//), 20
draw() method, 362
drawing
arcs, 364-365, 373
circles, 364
ellipses, 364
lines, 361-362
pie graphs, 365-366
PiePanel.java source code, 370
PieSlice class, 367-368
rectangles, 363
shapes, 361
drawRoundRect() method, 363-364
drawString() method, 157
E

EarthWeb’s Java directory, 416
Eclipse, 9
Android plug-in, 376. See also Android
installing, 422
projects, creating, 388
editing
NetBeans, 408-410
strings, 381
XML, 382
educational applications, 32
elements, 122
Ellipse2D class, 364
effices, drawing, 364
else statements, 95-96
employment opportunities, 417
emulators (Android), configuring, 382-384
enabling components, 233
encapsulation, 158
endless loops, 118
EndPoint class, 349
EndPoint Interfaces
  annotations, 346-348
  creating, 345
equal sign (=), 62, 65
equality operator (==), 93
equals() method, 82, 175
equalsIgnoreCase() method, 83
error handling, 277
  cannot resolve symbol message, 23
  catching exceptions, 277-278
    multiple exceptions, 282-284, 290
  PageCatalog sample application, 288-291
  try-catch blocks, 279-291
  try-catch-finally blocks, 284
  creating exceptions, 292
  ignoring exceptions, 287
  memory errors, 292
  stack overflows, 292
  throwing exceptions, 278, 284-286
  PageCatalog sample application, 288-291
  throw statements, 285
  try-catch statements, 303
  try...catch statements, 303
ErrorCorrectionModem class, 139
errors. See also error handling
  Android applications, 390
  arrayoutofbounds, 124
  bugs, 9
  exceptions, 124, 132
  javac error messages, 24
  logic errors, 9
  NetBeans, 411-412
  syntax errors, 9
escape codes, 79-80
evaluating expressions, 70
event handling, 227
  actionPerformed() method, 229, 305-306
  check boxes, 230
  combo boxes, 230
event listeners, 227-228
  ActionListener interface, 228
  LottoMadness application, 234-235, 238
  keyboard events, 232
event listeners, 227-228
  ActionListener interface, 228
  actionPerformed() method, 229
  adding, 227
  LottoMadness application, 234-238
EventListener interfaces, 227-228
Everlong.mp3 file, 333
Exception class, 278
exceptions, 124, 132
  ArrayIndexOutOfBoundsException
  Exception, 278
  catching, 277-278
  multiple exceptions, 282-284, 290
  PageCatalog sample application, 288-291
  try-catch blocks, 279-291
  try-catch-finally blocks, 284
  creating, 292
  ignoring, 287
  NumberFormatException, 281-282
  throwing, 278, 284-286
  PageCatalog sample application, 288-291
  throw statements, 285
executing applications, 8-9
existing objects, 177-178
exists() method, 331
exiting loops, 112
expressions, 59-60, 70-72. See also operators
  advantages, 71
  lambda. See closures
  operator precedence, 69-70
extends statement, 147, 175
extensions (file), .class, 26

F
File class, 330-331
FileInputStream class, 336
FileInputStream object, 339
FileOutputStream class, 336
File.pathSeparator, 330
files
  checking existence of, 331
  creating, 330
deleting, 331
File class, 331
file extensions, .class, 26
finding size of, 331
manifest, 391-392
reading streams, 331-334
renaming, 331
writing to streams, 336-337
fill() method, 362
fillRect() method, 361-363
fillRoundRect() method, 363
finding strings within strings, 84
Fisher, Timothy R., 413
flagging errors (NetBeans), 411
float statement, 61
floating-point variables,
  declaring, 61
FlowLayout layout manager, 196, 212
folders, 389. See also files
Font class, applying, 359-360
fonts, 272, 359
for loops, 108-110
  complex for loops, 114-115
  counter variables, 108
  empty sections, 115
  exiting, 112
  syntax, 108-110
formatting
  annotations, 346-348
  applications, 47-48, 217-222, 377-384
  behavior with methods, 157
  buttons, 195-196
  checkboxes, 198-199
  classes
    NetBeans, 408-410
    variables, 156
  Color class, 360
  combo boxes, 199
  components, 203-208
  Font class, 359-360
  inner classes, 313-320
  interfaces
    annotations, 346-348
    AWT, 189
    Endpoint Interfaces, 345
    labels, 197-198
    objects, 138, 146-149
    autoboxing/unboxing, 145-146
    converting, 141-142
panels, 203
Service Implementation Bean, 348-349
subclasses, 182-184
tables, 256-260
text
    areas, 200-202
    fields, 197-198
    threads, 296-297
    variables, 143-144, 153-155
web service clients, 353-355

formfeeds, 79
forward slash (/) character, 330
frames, 190
    adding components to, 195
    creating, 191
    SalutonFrame.java example, 194
    sizing, 192

G
Game application source code, 94
Gamelan website, 416
games, running on phones, 41
getActionCommand() method, 229, 239
getId() method, 397
getInsets() method, 217
ggetKeyChar() method, 231
ggetKeyCode() method, 231
ggetName() method, 331
ggetPort() method, 354
ggetProperty() method, 340
ggetSeconds() method, 158
ggetSource() method, 230, 249
ggetSquareRoot() method, 348, 352
ggetStateChange() method, 230
ggetTime() method, 348
ggetURL() method, 303
ggetValuesAdjusting() method, 249
ggetVirusCount() method, 166
Google, 41. See also Android
Gosling, James, 5, 31, 376, 405

graphics, 362
    applications, 233-242
    arcs, drawing, 364-365, 373
circles, drawing, 364
color, 359
    RGB values, 361
    setting, 361
    ellipses, drawing, 364
fonts, 359
icons, 252-255
    creating, 253
    Tool sample application, 254-256
lines, drawing, 362
PiePanel application, 365-366
PiePanel.java source code, 370
PieSlice class, 367-368
rectangles, drawing, 363

Graphics2D class, 362
    arcs, 364-365, 373
circles, 364
    ellipses, 364
    lines, 362
    rectangles, 363
graphs
    creating, 365-366
    PiePanel.java source code, 370
    PieSlice class, 367-368
    viewing, 372
greater than (>) operator, 93
greeting variable, displaying contents of, 21
GridLayout manager, 213-214
GridLayout() method, 221


GUIs (graphical user interfaces), 190, 247
    AWT, 189
    buttons, creating, 195-196
    change listeners, 249-250
    ColorSliders sample application, 252
    registering objects as, 249
    check boxes
        creating, 198-199
        event handling, 230
    combo boxes
        creating, 199-200
        event handling, 230
    event listeners, 227-228
    ActionListener interface, 228
    actionPerformed() method, 229
    adding, 227
    frames, 190
    adding components to, 195
    creating, 191
    sizing, 192
image icons, 252-255
    creating, 253
    Tool sample application, 254-256
Insets, 216-217
labels, creating, 197-198
layout managers, 211-213
    applications, 217-222
    BorderLayout, 214-216
    BoxLayout, 215
    FlowLayout, 212
    GridLayout, 213-214
    panels, creating, 203
    scroll panes, 201
        adding components to, 202
        creating, 201
        sliders, 248
    Swing, 189
tables, 256-260
text areas, 200
text fields
    creating, 197-198
    write-protecting, 223
toolbars, 252-253
    creating, 253-255
    dockable toolbars, 253
    Tool sample application, 254-256
windows, 190-195

H
handling errors. See error handling
Harwani, B.M., 414
hash maps, 269-272
Hemrajani, Anil, 414
hierarchies, Java classes, 173
history of Java, 31
HomePage.java listing, 288
horizontal sliders, 248
HttpComponents, 52
hyphen (-), 66

I
IceRocket, 416
icons, 252-255
    creating, 253
    Tool sample application, 254-256
ID3Reader application, 332
IDEs (integrated development environments), 9, 376, 405, 421
if-else statements, 95-96
if statements, 92-93, 104
blocks, 93-95
equal/not equal comparisons, 93
less than/greater than comparisons, 92-93
ignoring exceptions, 287
ImageIcon() method, 252-253
implementing Service Implementation Bean, 348-349
incrementing variables, 67-69
indexOf() method, 84
inequality operator (!=), 93
infinite loops, 118
InformIT, 414, 416
inheritance, 139, 150, 173-175
classes, 173-176
constructors, 160
hierarchy, 140-141
init() method, 303
initializing, definition of, 118
inner classes, 162-164
anonymous, 316-320
applying, 313-316
input/output. See I/O
Insets class, 216-217
installing
Android plug-ins, 423
Eclipse, 422
NetBeans, 405
tools, 10
int statement, 61
integers
arrays, creating, 122
variable types, 61
interpretation languages, 8, 12
interpreters, 33
definition of, 8
Java Plug-in, 33
I/O (input/output), 329
buffered input streams, 334-339
closing, 337
defining, 329-330
reading data from, 331-334
writing data to, 336-337
is statements, 91
ItemListener interface, 230
itemStateChanged() method, 230, 239
iteration, 109. See also loops
J
JApplet class, 173-174
inheritance, 174-175
methods
add(), 175
equals(), 175
overriding, 175
setBackground(), 175
setLayout(), 175
Java
classes, 409
documentation, 415
libraries, 51-54
educational applications, 32-33
history, 31
SourceForge, 39
Java Development Kits. See JDKs
The Java EE 6 Tutorial Basic Concepts, Fourth Edition, 413
Java Enterprise Edition. See JEE
Java Mobile Edition. See JME
Java Phrasebook, 413
Java Plug-in, 33
Java Standard Edition. See JSE
Java website, 415
javac command, 49
javac compiler, 23-24
JavaWorld, 34-35, 416
javax.xml.ws, 349
JAX-WS library packages, 354
JButton objects, 195-197
JCheckBox class, 198-199
JComboBox class, 199-200
JCompleteBox class, 199-200
JDKs (Java Development Kits), 10, 353
JEE (Java Enterprise Edition), 406
Jendrock, Eric, 413
JFrame class, 191
JLabel class, 197-198
JME (Java Mobile Edition), 406
job opportunities, 417
Joy, Bill, 31
JPanel class, 203
JScrollPane class, 201
JScrollPane() method, 201
JSE (Java Standard Edition), 406
JSlider class, 247
JSlider() method, 248
JTable component, 260. See also tables
JTextArea class, 200
JTextField class, 197-198
JToolBar() method, 253
JVMs (Java virtual machines), 33
K
keyboards
events, 230-232
input, monitoring, 320
KeyListener interface, 230-232
KeyView.java application, 232
methods, 153, 157
   actionPerformed(), 229, 239, 305-306
   add(), 175, 203
   addActionListener(), 228
   addChangeListener(), 249
   addComponent(), 253
   addItemListener(), 230
   addKeyListener(), 230
   addOneToField(), 238
   addSlice(), 368
   arguments, 159
   behavior, creating with, 157
   checkAuthor(), 164
   classes
      applying, 165-167
      declaring, 161
   clearAllFields(), 238
   close(), 337
   constructors, 159
      arguments, 160
      declaring, 160
      inheritance, 160
   createNewFile(), 331
   currentThread(), 305
   declaring, 157
   definition of, 82
   displaySpeed(), 138-139
   draw(), 362
   drawRoundRect(), 364
   drawString(), 157
   equals(), 82, 175
   equalsIgnoreCase(), 83
   exists(), 331
   fill(), 362
   fillRect(), 361, 363
   fillRoundRect(), 363
   get(), 270
   getActionCommand(), 229, 239
   getFileId(), 397
   getInsets(), 217
   getKeyChar(), 231
   getKeyCode(), 231
   getKeyText(), 231
   getName(), 331
   getPort(), 354
   getProperty(), 340
   getSeconds(), 158
   getSource(), 230, 249
   getSquareRoot(), 348, 352

loops
   array lists, 180-182
   benchmarks, 115-117
   definition of, 107
   do-while, 111-112
   exiting, 112
   for, 108-110
      complex for loops, 114-115
      counter variables, 108
      empty sections, 115
      syntax, 108-110
   infinite loops, 118
   naming, 113-114
   nesting, 113
   while, 110-111

LottoEvent.java class, 235, 238
LottoMadness application, 234
   applet version, 242
   event listeners, 234
LottoEvent.java class, 235-238
methods
   actionPerformed(), 239
   addOneToField(), 238
   clearAllFields(), 238
   getActionCommand(), 239
   itemStateChanged(), 239
   matchedOne(), 239
   numberGone(), 238
   source code listing, 240-242

lowercase, modifying strings, 83

M
magazines, JavaWorld, 34-35
main() blocks, Saluton program, 19
main() method, 408
MalformedURLException errors, 287
managers. See layout managers
managing
   applications, 93-95
   resources, 389-391
manifest files, Android applications, 391-392
maps, hash, 269-272
matchedOne() method, 239
Matz, Kevin, 8
memory errors, 292
messages
   errors. See errors
   SOAP, 354

lists
   arrays, 263-268
      looping, 180-182
      storing objects, 178-180
   choice lists, 230
load() method, 339
logic errors, 9
Long objects, 270
long variable type, 63
methods, 153, 157
   actionPerformed(), 229, 239, 305-306
   add(), 175, 203
   addActionListener(), 228
   addChangeListener(), 249
   addComponent(), 253
   addItemListener(), 230
   addKeyListener(), 230
   addOneToField(), 238
   addSlice(), 368
   arguments, 159
   behavior, creating with, 157
   checkAuthor(), 164
   classes
      applying, 165-167
      declaring, 161
   clearAllFields(), 238
   close(), 337
   constructors, 159
      arguments, 160
      declaring, 160
      inheritance, 160
   createNewFile(), 331
   currentThread(), 305
   declaring, 157
   definition of, 82
   displaySpeed(), 138-139
   draw(), 362
   drawRoundRect(), 364
   drawString(), 157
   equals(), 82, 175
   equalsIgnoreCase(), 83
   exists(), 331
   fill(), 362
   fillRect(), 361, 363
   fillRoundRect(), 363
   get(), 270
   getActionCommand(), 229, 239
   getFileId(), 397
   getInsets(), 217
   getKeyChar(), 231
   getKeyCode(), 231
   getKeyText(), 231
   getName(), 331
   getPort(), 354
   getProperty(), 340
   getSeconds(), 158
   getSource(), 230, 249
   getSquareRoot(), 348, 352

loops
   array lists, 180-182
   benchmarks, 115-117
   definition of, 107
   do-while, 111-112
   exiting, 112
   for, 108-110
      complex for loops, 114-115
      counter variables, 108
      empty sections, 115
      syntax, 108-110
   infinite loops, 118
   naming, 113-114
   nesting, 113
   while, 110-111

LottoEvent.java class, 235, 238
LottoMadness application, 234
   applet version, 242
   event listeners, 234
LottoEvent.java class, 235-238
methods
   actionPerformed(), 239
   addOneToField(), 238
   clearAllFields(), 238
   getActionCommand(), 239
   itemStateChanged(), 239
   matchedOne(), 239
   numberGone(), 238
   source code listing, 240-242

lowercase, modifying strings, 83

magazines, JavaWorld, 34-35
main() blocks, Saluton program, 19
main() method, 408
MalformedURLException errors, 287
managers. See layout managers
managing
   applications, 93-95
   resources, 389-391
manifest files, Android applications, 391-392
maps, hash, 269-272
matchedOne() method, 239
Matz, Kevin, 8
memory errors, 292
messages
   errors. See errors
   SOAP, 354

lists
   arrays, 263-268
      looping, 180-182
      storing objects, 178-180
   choice lists, 230
load() method, 339
logic errors, 9
Long objects, 270
long variable type, 63
methods

getStateChange(), 230
goingTime(), 348
goingURL(), 303
goingValuesAdjusting(), 249
getVirusCount(), 166
GridLayout(), 221
ImageIcon(), 252
indexOf(), 84
init(), 303
Intent(), 398
callStateChange(), 230, 239
JScrollPane(), 201
JSlider(), 248
JToolBar(), 253
Label(), 197
length(), 83, 331
listFiles(), 331
load(), 339
LottoMadness(), 221
main() blocks, 19
matchedOne(), 239
nextInt(), 53
numberGone(), 238
overriding, 175-176
pack(), 192
paint(), 176
parseInt(), 124, 169
println(), 72, 78, 157, 408
readLine(), 336
renameTo(), 331
return values, 87, 158
run(), 297, 304-305
setBackgroundColor(), 175
setContentView(), 397
setDefaultCloseOperation(), 192
setEditable(), 200, 223
setEnabled(), 233
setLayout(), 175, 212
setLayoutManager(), 195
setLookAndFeel(), 340
setSeconds(), 158
setSize(), 192
setText(), 244
setTitle(), 192
setVisible(), 193
shoot(), 269
showVirusCount(), 161
sleep(), 296

sort(), 127
sqrt(), 48
start(), 304
stateChanged(), 249
stop(), 301
substring(), 334
System.out.println(), 409
tauntUser(), 159
TextArea(), 200
toCharArray(), 124
toLowerCase(), 83
touppercase(), 83, 87
variables, scope, 161-162
void keyPressed(), 231
void keyReleased(), 231
void keyTyped(), 231
write(), 337

minus sign (-), 66-67

Modem class, 137-138
modifying strings, 83
modulus operator (%), 67
Monitor objects, 137
monitoring input, 320
mouse clicks, handling, 305-306
multidimensional arrays, 125-126
multiplication operator (*), 67
multitasking, 295
multithreading, 36, 295

N
naming

conventions

loops, 113-114
variables, 64, 74
file extensions, .class, 26
resources, 381

NASA, 36
navigating

Android applications, 379-380
programs, 8-9

nesting

classes, 162-164
loops, 113

NetBeans, 9

applying, 405
classes, 408-410
installing, 10, 405
projects, creating, 406-408
running, 410-411
troubleshooting, 411-412

NetBeans Field Guide, 405
NetBeansProjects, 407
New Android Project Wizard, 381
New File Wizard, 17
New Project Wizard, 407
new statement, 122, 159
newline characters (n), 79, 200
nextInt() method, 53
NumberFormatException, 281, 285
taukee() method, 238
numeric variable types, 62-63

0

Oak language, 31
object-oriented programming. See OOP

objects

array lists, 263-268
attributes, 136, 153
autoboxing/unboxing, 145-146
behavior, 136
casting, 143
classes, 136
closures, 321-326
converting, 141-144
creating, 138-149
creating, 177-178
hash maps, 269-272
inheritance, 139-141, 173-175
Long, 270
loops, 180-182
Modem, 137
Monitor, 137
PieChart, 136-137
Point, 268
runner, 309
sharing, 178
storing, 178-180
this statement, 164-165
variables

converting, 143-144
declaring, 153-155
private, 155
protected, 155

onCreate() method, 397
online communities, Stack Overflow, 416
OOP (object-oriented programming), 136, 190
  advantages, 136-138
  applications, debugging, 137
  encapsulation, 158
  inheritance, 139-141, 150, 173-175
  objects
    casting, 143
    creating, 138-139, 146-149
operators
  +=, 81
  addition (+), 66
  concatenation (+), 80-81
  decrement (−), 67
  division (/), 66
  equality (==), 93
  greater than (>), 93
  inequality (!=), 93
  less than (<), 92-93
  modulus (%), 67
  multiplication (*), 67
  precedence, 69-70
  subtraction (−), 66
  ternary (?), 98
Oracle, 5, 29
Oracle Technology Network for Java Developers, 414
order of precedence, operators, 69-70
organizing resources, 389-391.
See also managing output. See I/O
@Override annotation, 347
overriding methods, 175-176
P
pack() method, 192
Package Explorer, applying, 380
packages, 155
  Android SDKs, installing, 423
  javax.xml.ws package, 349
  JAX-WS library, 354
PageCatalog application, 288-291
pageTitle array, 302
paint() method, overriding, 176
panels, creating, 203
parseInt() method, 144, 169
passing arguments to methods, 159
pasting strings, 80-82
percent sign (%), 67
percentages, calculating, 206
performance, interpreted languages, 12
phones
  Android, configuring, 424-426
  Java, running on, 41
PHP, 5
Pie graphs
  creating, 365-370
  viewing, 372
PieChart object, 136-137
PieFrame application, 371-372
PiePanel application, 365-370
  PiePanel.java source code, 370
  PieSlice class, 367-368
PieSlice class, 367-368
pipe (|) characters, 282
PlanetWeight application, 71
platform independence, 33
Playback.java, 196
plug-ins
  Android, 376
  installing, 423
plus sign (+)
  addition operator (+), 66
  concatenation operator, 80-81
  increment operator (++), 67
Point class, 182, 268
Point3D class
  creating, 183-184
  testing, 184
postfixing, 67
precedence, operators, 69-70
prefixing, 67
printing strings, special characters, 79-80
println() method, 72, 78, 157, 408
private classes, 150
private variables, 155
procedures, System.out.println(), 72
program listings. See code listings
programming. See also code; languages
  Android, 421
    configuring phones, 424-426
    Eclipse, 422
    plug-ins, 423
OOP. See also OOP
  advantages of, 136-137
  overview of, 135
    tools
      installing, 10
      selecting, 9
programs. See also applications
  creating, 47-48
  running, 24-25
  Saluton, 16-25
  starting, 16-18
  strings, viewing in, 78
  TextDisplayer, 49
  troubleshooting, 23
  writing, 15-16
Project Location text field, 407
Project Selection dialog box, 384
projects
  Android applications, 379-380
  creating, 388
  NetBeans, 406-408
properties
  configuration, reading/writing, 339-342
  customizing, 395
Properties object, 340
protected variables, 155
public methods, 158
public statements, 138
publishing web services, 349-350
Python, 4
Q
QName, 353
question mark (?), 98
quotation marks
  double ("), 61
  escape codes, 79
  single (‘), 61
QuoteMedia, 37-38
R
R class, 397
Random class, 55
Read Console application, 335
reading
  configuration properties, 339-342
  files, 331-334
readLine() method, 336
ReadyBASIC interpreter, 8
real-world Java projects
  JavaWorld website, 34-35
recommended reading, 413
Rectangle2D class, 363
rectangles, drawing, 363
Red, Green Blue (RGB) color system, 361
referencing objects, this statement, 164-165
registering objects as change listeners, 249
renameTo() method, 331
renaming files, 331
resources
  Android, 391
  folders, viewing, 389
  Java-related books, 413
  job opportunities, 417
  managing, 389-391
  naming, 381. See also websites strings, editing, 381
restricting access, 155. See also access control
return values (methods), 87, 158
Revolve applet, 301
RGB (red, green, blue) color system, 361
R.java file, 397
Root application
  compiling, 49
  source code, 48
rounded rectangles, drawing, 363
Ruby, 4
Run, Run Main Project, 51
run() method, 297, 304-305
RuneScape, 30
Runnable interface, 295
runner objects, 309
running
  Android applications, 385-387
  Java on phones, 41
  NetBeans, 406-411
  programs, 24-25
  threads, 304-305
S
Saluton application, 16-18
  classes, 18-19
  code code listings, 22
  compiling, 22-23
  main() block, 19
  saving, 21
  variables, 21
  writing, 16-25
  SalutonFrame.java, 194-195
Sams Teach Yourself Java 2 in 21 Days, 413
Sams Teach Yourself Java 2 in 24 Hours website, 419-420
Sams Teach Yourself Java in 24 Hours website, 415
saving
  applications, 8
  Saluton program, 21
scope (variables), 161-162
scroll panes, 201-202
SDKs (Software Development Kits), 421
searching strings, 84
security, 35-36
selecting
  languages, 4-5
  tools, 9
semicolon (;), 20, 26, 114
sending arguments to applications, 49-51
Service Implementation Bean, 348-349
services
  clients, creating, 353-355
  defining, 345
  publishing, 349-350
  SquareRootServer, 345
setBackground() method, 175
setContentView() method, 397
setDefaultCloseOperation() method, 192
setEnabled() method, 200, 223
setEditable() method, 233
setLayout() method, 175, 212
setLayoutManager() method, 195
setLookAndFeel() method, 194
setProperty() method, 340
setSeconds() method, 158
setSize() method, 192
setText() method, 244
setTitle() method, 192
setVisible() method, 193
shapes
  arcs, 364-365, 373
circles, 364
drawing, 361-362
  ellipses, 364
d lines, 362
PiePanel application, 365-366
  PiePanel.java source code, 370
  PieSlice class, 367-368
  rectangles, 363
sharing objects, 178
shoot() method, 269
short variable type, 62
showVirusCount() method, 161
signatures (digital), 35
simple variables, converting, 141-142
single quotation marks (‘), 79
skills, language, 4-5
SkyWatch, 36
Slashdot, 415
slashes (/\), 20
sleep() method, 296
sliders, 247-248
slowing down threads, 296
SOAP messages, 354
software. See applications; programs
  Software Development Kits. See SDKs
sort() method, 127
sorting arrays, 126-127
source code listings. See code listings
SourceForge, 39
sources (casting), 141
spacing in source code, 26
Spartacus.java class, 409
special characters, escape codes, 79-80
sqrt() method, 48
square brackets ([]), 122
SquareRootClient application, 348-349
SquareRootServer application, 348
SquareRootServer web service, 345
SquareRootServerPublisher application, 349
sRGB (Standard RGB), 361
stack overflows, 292, 416
Standard RGB, 361
start() method, 304
starting
  programs, 16-18
  threads, 304
  variables, 65
stateChanged() method, 249
statements, 59
benchmarks, 115-117
blocks, 19-20, 59
braces ({}) notation, 20
break, 97, 104, 112
case, 96
catch, 310
class, 18-19, 138
continue, 113
default, 97
definition of, 6
equation, 7
expressions, 60, 70-72
extends, 147, 175
float, 61
if, 91-92, 104
blocks, 93-95
equal/not equal comparisons, 93
less/greater than comparisons, 92-93
if-else, 95-96
int, 61
loops
definition of, 107
do-while, 111-112
exiting, 112
for, 108-115
infinite loops, 118
naming, 113-114
nesting, 113
while, 110-111
new, 122, 159
public, 138
static, 156, 161
super, 177, 183
switch, 96
this, 164-165, 176, 183
throw, 285
try-catch blocks, 279-284, 290-291, 303
try-catch-finally blocks, 284
void, 157
static statements, 156-161
static variables, 156
stop() method, 301
stopping threads, 309
storage
arrays, 121
applying, 123-125
declaring, 122-123
multidimensional, 125
sorting, 126-127
fonts, 272
objects, 178-182
array lists, 263-268
hash maps, 269-272
text in strings, 77-78
variables, 20, 65-66
streams, 329-330
buffered input streams, 334-339
Console application, 336
ReadConsole application, 335
closing, 337
defining, 329-330
reading from, 331-334
writing to, 336-337
String data type, 20
strings, 77-78
adding to, 81-82
arrays, 122
case, modifying, 83, 87
characters, counting, 128-130
comparing, 82
concatenating, 80
definition of, 61, 77
equal/not equal comparisons, 93
finding within other strings, 84
length, determining, 83
less/greater than comparisons, 92-93
programs, viewing in, 78
resources, editing, 381
searching, 84
special characters, 79-80
text, 77-78
variables, 61-62
declaring, 78
linking, 81-82
strings.xml file, 382
Stroustrup, Bjarne, 5
subclasses, 140, 148, 175-184
substring() method, 334
subtraction operator (-), 66
Sun website, 29-31, 414-415
super statement, 177, 183
superclasses, 140
Swing, 189, 247
buttons, creating, 195-196
change listeners, 249-250
ColorSliders sample application, 252
registering objects as, 249
check boxes
creating, 198-199
event handling, 230
combo boxes
creating, 199-200
event handling, 230
documentation, 261
enabling/disabling components, 233
event listeners, 227-228
ActionListener interface, 228
actionPerformed() method, 229
adding, 227
LottoMadness application, 234-238
image icons, 252-255
creating, 253
Tool sample application, 254-256
labels, creating, 197-198
layout managers, 211-213
applications, 217-222
BorderLayout, 214-216
BoxLayout, 215
FlowLayout, 212
GridLayout, 213-214
panels, creating, 203
scroll panes, 201
adding components to, 202
creating, 201
sliders, 248
creating, 248
labels, 248
text areas, creating, 200
text fields
creating, 197-198
write protecting, 223
toolbars, 252-253
creating, 253-255
dockable toolbars, 253
Tool sample application, 254-256
switch statements, 96
syntax errors, 9
System.out.println() method, 409
System.out.println() procedure, 72

T

tables, 256-260
tabs, escape code, 79
tauntUser() method, 159
ternary operator (?), 98
testing
  computer speed, 115-117
  Points3D class, 184
  SquareRootServerPublisher
  application, 351
text. See also strings
  areas, 200
  Color class, 360
  editors, 8
  fields, 197-198, 223
  Font class, 359-360
  pasting into strings, 82
  strings
    pasting, 81
    storage, 77-78
TextArea() constructor method, 200
TextDisplayer program, 49
this statement, 164-165, 176, 183
Thread class, 295
threads, 295
  applets, 301
  classes, 296-300
  creating, 296-300
  multithreading, 36
  Runnable interface, 295
  running, 304-305
  slowing down, 296
  starting, 304
  stopping, 309
  Thread class, 295
throw statements, 285
throwing exceptions, 278, 284-286
  PageCatalog sample application, 288-291
  throw statements, 285
time, displaying, 410	
titles, frames, 192
T-Mobile G1s, 375
toCharArray() method, 124
toLowerCase() method, 83
Tool application, 254-256
toolbars, 252-253
  creating, 253-255
  docking, 256
  Tool sample application, 254-256
tools
  development, 4
  installing, 10
  selecting, 9

toUpperCase() method, 83, 87
troubleshooting. See also errors
  Android applications, 390
  applications, 9, 23
  BlankFiller.java, 50
  exceptions, 277-279. See also exceptions
  NetBeans, 411-412
trusted developers, 35
try-catch blocks, 279-284, 290-291
  DivideNumbers sample application, 290
  NumberDivider sample application, 284
  SumNumbers sample application, 279-281, 290-291
try-catch statement, 303
try-catch-finally blocks, 284
TryPoints.java listing, 184
Twitter, 417
two slash characters (/\), 287
types
  Boolean, 63-64
  byte, 62
  char, 61-62
  long, 63
  short, 62
  streams, 330

type values (variables), casting, 142

U
Udovydchenko, Aleksey, 40
unboxing, 145-146
underscore (_), 64
University of British Columbia, 32
upper limits of arrays, checking, 124
uppercase, modifying, 83, 87
user events, 227
  ActionListener interface, 228
  combo boxes, 230
  components, enabling/disabling, 233
  handling, 229
  keyboard events, 230-232
  LottoMadness application, 233-239

V
van de Panne, Michiel, 32
Variable application
  int statement, 60
variables
  floats, 60
  integer, 61
  integers, 61
  strings, 61-62
variables
  access control, 155
  applying, 165-167
arrays
  applying, 123-125
  declaring, 122-123
  definition of, 121
  initial values, 122
  multidimensional, 125
  sorting, 126-127
  casting, 142
  characters, 61-62
  classes, 156
  configuring, 302
  converting, 141-142
  converting to objects, 144
  counter variables
    definition of, 108
    initializing, 108
  creating, 153-155
  data types, 20
  declaring, 60
  definition of, 59
  displaying contents of, 21
  initializing, 118
  length, 132
  naming conventions, 64, 74
  objects, converting, 143-144
  private, 155
  protected, 155
  scope, 161-162
  storage, 20
  strings, 78
    changing case, 87
    comparing, 82
    concatenating, 80
    declaring, 78
    determining length, 83
    escape codes, 79
    linking, 81-82
    modifying case, 83
    viewing, 78
XML (Extensible Markup Language), editing

W
Web Service Description Language, See WSDL

Web services
- clients, creating, 353-355
- publishing, 349-350
- SquareRootServer, 345

weblogs, 416
@WebMethod annotation, 347

websites
- Gamelan, 416
- InformIT, 414
- JavaWorld, 34-35
- Sams Publishing, 414
- Sams Teach Yourself Java 2 in 24 Hours, 419-420
- Sams Teach Yourself Java in 24 Hours, 415
- Slashdot, 415
- Sun, 29, 31, 414-415
- VeriSign, 36
- Workbench, 415

Wheel of Fortune application, 129

while loops, 110-112

widgets, customizing, 395

windows, 190-195, 384

Windows, compiling Java applications, 23

wizards
- New Android Project Wizard, 381
- New File, 17
- New Project Wizard, 407

word processing programs, 8

Workbench website, 415

write() method, 337

write protecting text fields, 223

writing
- Android apps, 375-377
- applications, 15-25, 47-48
- code, 396-401
- Color class, 360
- configuration properties, 339-342
- Font class, 359-360
- programs, 5
- streams, 336-337

WSDL (Web Service Description Language), 351, 353

X
XML (Extensible Markup Language), editing, 382

this statement, 164-165

types
- assigning, 60
- Boolean, 63-64
- char, 61-62, 77
- floating-point, 61
- integers, 61
- long, 63
- short, 62
- strings, 61-62

values
- assigning, 65-66
- decrementing, 67-69
- incrementing, 67-69
- starting values, 65

VB.NET, 5

VeriSign website, 36

versions, Java, 31

vertical sliders, creating, 248

viewing
- Android projects, 379
- credits, 84-86
- pie graphs, 372
- resources, 389

strings
- in programs, 78
- special characters, 79-80
- text areas, 200

web services, 355

Virus application, 165

- class constructor, 160

methods
- getSeconds(), 158
- setSeconds(), 158
- tauntUser(), 159
- showVirusCount(), 161

Virus class, 153

VirusLook application source code, 166

void keyPressed() method, 231

void keyReleased() method, 231

void keyTyped() method, 231

void statement, 157