Minecraft Mods Programming

ABSOLUTE BEGINNER’S GUIDE

No experience necessary!

Rogers Cadenhead

FREE SAMPLE CHAPTER

SHARE WITH OTHERS
Minecraft Mods Programming

ABSOLUTE BEGINNER’S GUIDE

Rogers Cadenhead
Contents at a Glance

Part I  Java from the Ground Up
1  Dig Into Minecraft Programming with Java .......................... 1
2  Use NetBeans for Minecraft Programming ............................ 15
3  Create a Minecraft Mod .................................................. 25
4  Start Writing Java Programs ............................................ 37
5  Understand How Java Programs Work ................................. 51
6  Store and Change Information in a Mod ............................... 63
7  Use Strings to Communicate ............................................. 79
8  Use Conditional Tests to Make Decisions ............................. 91
9  Repeat an Action with Loops ............................................ 107
10 Store Information with Arrays ........................................... 119

Part II  The World of Java Objects
11 Create Your First Object ................................................. 131
12 Describe What Your Object Is Like .................................... 147
13 Make the Most of Existing Objects .................................... 163
14 Store Objects in Data Structures ....................................... 177
15 Handle Errors in a Mod ................................................... 189
16 Create a Threaded Mod .................................................... 205
17 Read and Write Files ...................................................... 223

Part III  Create Killer Minecraft Mods
18 Spawn a Mob ....................................................................... 241
19 Make One Mob Ride Another ............................................ 257
20 Take a Census of Mobs and Villages ................................. 267
21 Transmute Materials in an Inventory ................................. 281
22 Dig a Giant Hole ........................................................... 293
23 Chop Down a Forest of Trees ............................................ 307
24 Respond to Events in the Game ....................................... 321
25 Display a Mob’s Health During Combat ............................ 335
26 Make a World Change Over Time ..................................... 349
27 Befriend the God of Lightning ......................................... 359
A  Visit This Book’s Website .................................................. 371
Index .................................................................................. 373
Table of Contents

1 Java from the Ground Up

1 Dig Into Minecraft Programming with Java .................................................... 1
   Setting Up a Minecraft Server ........................................................................ 2
   Fixing Problems Running the Server ................................................................ 5
   Connecting to the Server .............................................................................. 9
   Fixing a Server Connection Problem ............................................................ 10

2 Use NetBeans for Minecraft Programming .................................................. 15
   Installing NetBeans ...................................................................................... 16
   Creating a New Project .................................................................................. 16
   Creating a New Java Class ........................................................................... 18
   Running the Application .............................................................................. 21
   Fixing Errors ................................................................................................. 22

3 Create a Minecraft Mod ................................................................................. 25
   Creating Your First Mod .............................................................................. 26

4 Start Writing Java Programs ........................................................................... 37
   What You Need to Write Programs ............................................................... 38
   Creating the Splash Program ....................................................................... 38
   Beginning the Program .................................................................................. 40
   The class Statement ..................................................................................... 42
   What the main Statement Does ..................................................................... 42
   Those Squiggly Bracket Marks ..................................................................... 43
   Storing Information in a Variable ................................................................... 44
   Displaying the Contents of a Variable ......................................................... 44
   Saving the Finished Product ......................................................................... 45
   Compiling the Program into a Class File ..................................................... 46
   Fixing Errors ................................................................................................. 46
### 5 Understand How Java Programs Work

- Creating an Application ............................................. 52
- Sending Arguments to Applications ............................. 54
- The Java Class Library ................................................. 56

### 6 Store and Change Information in a Mod

- Statements and Expressions ........................................ 64
  - Assigning Variable Types ........................................... 64
    - Integers and Floating-Point Numbers ......................... 65
    - Characters and Strings ........................................... 65
    - Other Numeric Variable Types .................................. 67
    - The boolean Variable Type ................................. 68
  - Naming Your Variables ............................................. 69
  - Storing Information in Variables ................................ 70
  - All About Operators ................................................ 71
    - Incrementing and Decrementing a Variable ................. 72
    - Operator Precedence ............................................ 74
  - Using Expressions .................................................. 75

### 7 Use Strings to Communicate

- Storing Text in Strings .............................................. 80
- Displaying Strings in Programs ................................... 80
- Using Special Characters in Strings ............................ 81
- Pasting Strings Together ............................................ 82
- Using Other Variables with Strings ............................ 83
- Advanced String Handling ......................................... 84
  - Comparing Two Strings ........................................... 84
  - Determining the Length of a String ........................... 85
  - Changing a String’s Case ......................................... 85
  - Looking for a String .............................................. 86
- Presenting Credits ..................................................... 87
Building an Inheritance Hierarchy .................................................. 135
Converting Objects and Simple Variables ........................................ 136
   Casting Simple Variables ...................................................... 137
   Casting Objects ...................................................................... 138
   Converting Simple Variables to Objects and Back ...................... 139
   Autoboxing and Unboxing ...................................................... 141
Creating an Object ........................................................................ 142

12 Describe What Your Object Is Like ............................................. 147
Creating Variables ........................................................................ 148
Creating Class Variables ................................................................ 150
Creating Behavior with Methods .................................................... 151
   Declaring a Method ................................................................ 151
   Similar Methods with Different Arguments .............................. 153
   Constructor Methods ................................................................ 154
   Class Methods ...................................................................... 155
   Variable Scope Within Methods ............................................. 155
Putting One Class Inside Another ................................................... 157
Using the this Keyword .................................................................. 158
Using Class Methods and Variables ................................................. 159

13 Make the Most of Existing Objects .............................................. 163
   The Power of Inheritance ...................................................... 164
   Inheriting Behavior and Attributes ........................................... 165
   Overriding Methods ................................................................ 165
   Establishing Inheritance ....................................................... 165
   Using this and super in a Subclass ......................................... 166
Working with Existing Objects ....................................................... 167
Storing Objects of the Same Class in Array Lists .............................. 168
   Looping Through an Array List .............................................. 170
Creating a Subclass ....................................................................... 172

14 Store Objects in Data Structures ................................................. 177
   Array Lists ............................................................................ 178
   Hash Maps ............................................................................ 184
   Why Classes Are Synchronized ............................................. 188
15 Handle Errors in a Mod ................................................................. 189
  Exceptions .................................................................................. 190
  Catching Exceptions in a try-catch Block .................................. 191
  Catching Several Different Exceptions ..................................... 194
  Handling Something After an Exception ................................. 196
  Throwing Exceptions ................................................................ 197
  Ignoring Exceptions .................................................................. 199
  Exceptions That Don’t Need Catch .......................................... 200
  Throwing and Catching Exceptions ........................................... 201

16 Create a Threaded Mod ............................................................... 205
  Threads ...................................................................................... 206
  Slowing Down a Program .......................................................... 206
  Creating a Thread ..................................................................... 207
  Working with Threads .............................................................. 211
    The class Declaration ............................................................... 212
    Setting Up Variables ............................................................... 212
  The Constructor ....................................................................... 213
  Catching Errors as You Set Up URLs .................................... 213
  Starting the Thread .................................................................. 214
    Running the Thread ................................................................. 215
  Handling Mouse Clicks ............................................................ 216
    Displaying Revolving Links .................................................... 217
    Stopping a Thread .................................................................. 220

17 Read and Write Files ................................................................. 223
  Streams ...................................................................................... 224
    Files ....................................................................................... 225
      Reading Data from a Stream ............................................... 226
      Buffered Input Streams ....................................................... 229
  Writing Data to a Stream .......................................................... 232
  Reading and Writing Configuration Properties ....................... 235
### III Create Killer Minecraft Mods

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>Spawn a Mob</td>
<td>241</td>
</tr>
<tr>
<td></td>
<td>The Mod Framework</td>
<td>242</td>
</tr>
<tr>
<td></td>
<td>Starting a Mod Project</td>
<td>245</td>
</tr>
<tr>
<td></td>
<td>Writing the Mod's Code</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td>Deploying a Mod</td>
<td>254</td>
</tr>
<tr>
<td>19</td>
<td>Make One Mob Ride Another</td>
<td>257</td>
</tr>
<tr>
<td></td>
<td>Starting the Project</td>
<td>258</td>
</tr>
<tr>
<td></td>
<td>Writing the Mod</td>
<td>259</td>
</tr>
<tr>
<td></td>
<td>Deploying the Mod</td>
<td>264</td>
</tr>
<tr>
<td>20</td>
<td>Take a Census of Mobs and Villages</td>
<td>267</td>
</tr>
<tr>
<td></td>
<td>Starting the Project</td>
<td>268</td>
</tr>
<tr>
<td></td>
<td>Creating the Project</td>
<td>269</td>
</tr>
<tr>
<td></td>
<td>Deploying the Mod</td>
<td>278</td>
</tr>
<tr>
<td>21</td>
<td>Transmute Materials in an Inventory</td>
<td>281</td>
</tr>
<tr>
<td></td>
<td>Starting the Project</td>
<td>282</td>
</tr>
<tr>
<td></td>
<td>Creating the Project</td>
<td>284</td>
</tr>
<tr>
<td></td>
<td>Deploying the Mod</td>
<td>289</td>
</tr>
<tr>
<td>22</td>
<td>Dig a Giant Hole</td>
<td>293</td>
</tr>
<tr>
<td></td>
<td>Starting the Project</td>
<td>294</td>
</tr>
<tr>
<td></td>
<td>Creating the Project</td>
<td>295</td>
</tr>
<tr>
<td></td>
<td>Deploying the Mod</td>
<td>301</td>
</tr>
<tr>
<td></td>
<td>Backing Up a Minecraft World</td>
<td>302</td>
</tr>
<tr>
<td>23</td>
<td>Chop Down a Forest of Trees</td>
<td>307</td>
</tr>
<tr>
<td></td>
<td>Starting the Project</td>
<td>308</td>
</tr>
<tr>
<td></td>
<td>Creating the Project</td>
<td>309</td>
</tr>
<tr>
<td></td>
<td>Deploying the Mod</td>
<td>318</td>
</tr>
<tr>
<td>24</td>
<td>Respond to Events in the Game</td>
<td>321</td>
</tr>
<tr>
<td></td>
<td>Starting the Project</td>
<td>322</td>
</tr>
<tr>
<td></td>
<td>Creating the Project</td>
<td>323</td>
</tr>
<tr>
<td></td>
<td>Deploying the Mod</td>
<td>331</td>
</tr>
<tr>
<td>Chapter</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>25</td>
<td>Display a Mob’s Health During Combat</td>
<td>335</td>
</tr>
<tr>
<td></td>
<td>Starting the Project</td>
<td>336</td>
</tr>
<tr>
<td></td>
<td>Creating the Project</td>
<td>338</td>
</tr>
<tr>
<td></td>
<td>Deploying the Mod</td>
<td>345</td>
</tr>
<tr>
<td></td>
<td>Learning Bukkit Methods from NetBeans</td>
<td>346</td>
</tr>
<tr>
<td>26</td>
<td>Make a World Change Over Time</td>
<td>349</td>
</tr>
<tr>
<td></td>
<td>Starting the Project</td>
<td>350</td>
</tr>
<tr>
<td></td>
<td>Creating the Project</td>
<td>351</td>
</tr>
<tr>
<td></td>
<td>Deploying the Mod</td>
<td>357</td>
</tr>
<tr>
<td>27</td>
<td>Befriend the God of Lightning</td>
<td>359</td>
</tr>
<tr>
<td></td>
<td>Starting the Project</td>
<td>360</td>
</tr>
<tr>
<td></td>
<td>Stepping Through Mod Development</td>
<td>360</td>
</tr>
<tr>
<td></td>
<td>Creating the Project</td>
<td>361</td>
</tr>
<tr>
<td></td>
<td>Deploying the Mod</td>
<td>366</td>
</tr>
<tr>
<td></td>
<td>Joining the Community of Mod Developers</td>
<td>367</td>
</tr>
<tr>
<td>A</td>
<td>Visit This Book’s Website</td>
<td>371</td>
</tr>
</tbody>
</table>

Index ........................................................................................................ 373
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 24 Hours. 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.javaminecraft.com.

Dedication

This book is dedicated to the kids out there who have been inspired by Minecraft to learn computer programming, whether they’re 10, 20, or 50. There are a lot of great experiences ahead of you, not only in writing mods for a video game but in what you do with your skills beyond the game world.

Acknowledgments

To the folks at Pearson, especially Mark Taber, Tonya Simpson, Boris Minkin, and Megan Wade-Taxter. 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.

Email: feedback@quepublishing.com

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

Reader Services

Visit our website and register this book at quepublishing.com/register for convenient access to any updates, downloads, or errata that might be available for this book.
This page intentionally left blank
IN THIS CHAPTER

• Type a Java program into a source code 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

START WRITING JAVA PROGRAMS

A computer program is a set of instructions that tells a computer what to do. These instructions are given to a computer using a programming language.

During this chapter, you create a simple program with the Java language by entering it into a text editor. When that’s done, you save the program, compile it, and test it. Then you break it on purpose and fix it again, just to show off.
What You Need to Write Programs

To create Minecraft mods or any other 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.

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 Notepad or Text Edit and make changes, and then open it again later in NetBeans without any problems.

Creating the Splash Program

One of the funny quirks of Minecraft is the random message that displays in the game client when the program is run. It appears as yellow text across a corner of the Minecraft logo, as shown in Figure 4.1.
CHAPTER 4  START WRITING JAVA PROGRAMS  39

The splash in Figure 4.1 is “Don’t feed avocados to parrots!” Mojang, the developer of Minecraft, uses the splash to crack jokes, make references to classic or obscure videogames, and say other unusual things. Sometimes it even dispenses good advice, as in this particular example. Avocados are poisonous to birds and harmful to many other animals.

The first Java program that you create will have its own simple splash message: “Blue warrior shot the food!”

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

2. Select the project category Java and the project type Java Application; then click Next.
3. Enter Minecraft 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 Minecraft project is created in its own folder. You can use this project for the Java programs you write as you progress through this book—at least as far as Chapter 17, “Read and Write Files.” After that, you will be creating each Minecraft mod as its own project in NetBeans.

**Beginning the Program**

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

2. Find and select the folder where you installed the Bukkit server (if necessary).
3. Open that folder.
4. Select Minecraft and click Open Project.

The Minecraft 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 the project contains.

To add a new Java program to the currently open project, select File, New File. The New File Wizard opens, as shown in Figure 4.2.

![New File Wizard](Image)

**FIGURE 4.2**

The New File Wizard.
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, select 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 `Splash`.
2. In the Package field, enter `com.javaminecraft`.
3. Click Finish.

So you can begin working right away on your program, an empty file named `Splash.java` opens in the source code editor. Using the editor, begin your Java programming career by entering each line from Listing 4.1. These statements are called the program’s source code.

**CAUTION** As you were warned in an earlier chapter, 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 4.1  The Splash Program**

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

Be 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, select File, Save to save the file.

At this point, `Splash.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 `Splash` 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, aside from the sample Minecraft mod you wrote and tested in Chapter 3, “Create a Minecraft Mod.”
The class Statement

The first line of the program is the following:

```java
package com.javaminecraft;
```

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

After a blank line, the third line is this:

```java
class Splash {
```

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

As you might recall from Chapter 3, each instruction you give a computer is called a statement. The `class` keyword 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 `Splash` matches the document’s filename, `Splash.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. Mods are an exception because they are run by the Bukkit server and cannot be run directly. When a player types a command that a mod supports, the server runs that mod.

Some other 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.

The Java programs you write during the next 13 chapters use `main` as their starting point. That’s because you run them directly on your computer. Mods, applets, apps, and servlets are run indirectly by another program or device.
To differentiate programs with `main` from these other types, they are called `applications`.

**Those Squiggly Bracket Marks**

In the `Splash` 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 4.1, 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 `Splash` 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.

**TIP** NetBeans can help you figure out where a block begins and ends. Click one of the brackets in the source code of the `Splash` program. The bracket you clicked turns yellow along with its corresponding bracket. The Java statements enclosed within the two yellow brackets are a block. This tip is not that useful on a short program like `Splash`, but as you write much longer programs, it helps you avoid looking like a blockhead.

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

```
// 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.
NOTE  Semicolons are required at the end of each Java statement, but in the Splash program the line // My first Java program goes here does not end with a semicolon.

The reason this is permitted is because 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:

```java
System.out.println(greeting); // Exploding creepers!
```

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 Splash.java file, replace Line 5 with the following:

```java
String greeting = "Blue warrior shot the food!";
```

This statement tells the computer to store the text Blue warrior shot the food! 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 Splash program after the String greeting = "Blue warrior shot the food!" 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 4.2, although you might have used slightly different spacing in Lines 5–6. Make any corrections that are needed and save the file (by selecting the menu command File, Save).

#### LISTING 4.2  The Finished Version of the Splash Program

```java
1: package com.javaminecraft;
2: class Splash {
3:     public static void main(String[] arguments) {
4:         String greeting = "Blue warrior shot the food!";
5:         System.out.println(greeting);
6:     }
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 4.3 shows what the program would look like if it was written in the English language instead of Java.

#### LISTING 4.3  A Line-by-Line Breakdown of the Splash Program

```java
1: Put this program in the com.javaminecraft package.
2: class Splash {
3:     public static void main(String[] arguments) {
4:         String greeting = "Blue warrior shot the food!";
5:         System.out.println(greeting);
6:     }
7: }
```
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 4.2, the program compiles successfully.

A compiled version of the program, a new file called `Splash.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 `Splash` only one class is needed.

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

**NOTE**  The Java compiler speaks up only when there’s an error to complain about. If you compile a program successfully without any errors, nothing happens in response. This is disappointing. When I was starting out as a Java programmer, I was hoping successful compilation would be met with a grand flourish of celebratory horns.

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 4.3.

```java
6:   Display the contents of the variable greeting
7:     The main part of the program ends here.
8: The Splash program ends here.
```
FIGURE 4.3
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 Splash program:

cannot find symbol.
symbol : variable greting
location: class Splash

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?

System.out.println(greeting);
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 Splash program, double-check that your program matches Listing 4.2 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 needs to be fixed.

Take note that 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.

**TIP** This book’s official website at www.javaminecraft.com includes source files for all programs you create. If you can’t find any typos or other reasons for errors in the Splash program but there are still errors, go to the book’s website and download `Splash.java` from the Chapter 4 page. Try to run that file instead.

**Running a Java Program**

To see whether the Splash program does what you want, run the class with the JVM, the interpreter that runs all Java code. In NetBeans, select 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 4.4.
If you see the text Blue warrior shot the food!, you have written, compiled, and run the Java program successfully.

NOTE The message Blue warrior shot the food! is a splash message in Minecraft that pays homage to Gauntlet, a 1980s coin-operated videogame in which up to four players roamed through a dungeon killing monsters and accumulating treasure. Because players needed food to stay alive, accidentally shooting the food was a major faux pas.

You can find all of Minecraft’s known splash messages and explanations for many of them on Minecraft Wiki. Visit the web page at http://minecraft.gamepedia.com/Splash.

Blank Spaces and Whitespace in a Java Program

As you typed in the Splash program in Listing 4.1, taking care to get the right number of spaces before each statement to make sure it all lined up properly, you might have asked yourself whether that’s important. Blank spaces and whitespace are 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 doesn’t care. You could have written the Splash 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.

**THE ABSOLUTE MINIMUM**

During this chapter, you got an introduction to all the elements of a Java program. You learned that to develop a program you need to complete these three 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.

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 chapters. As long as you got the Splash program to work during this chapter, you’re ready to proceed.
Index
integers
declaring, 120
Wheel of Fortune
application, 129
lists
adding objects, 169
creating, 168, 178
declared, 178
elements, retrieving, 169
generics, 178
looping through, 170
objects. See objects, array lists
outside-the-cube trees, 310-311
points, creating, 180-183
size, 179
vectors, compared, 179
multidimensional, 124-125
pageLink, 213
pageTitle, 212-213
sample application, 123-124
searched locations,
storing, 313
sorting, 125-126
strings, 120
upper limits, 122
Wheel of Fortune
application, 127-130
Arrays class, 125
asterisk (*) (multiplication
operator), 72
attributes, 132
defined, 147
inheritance, 135
autoboxing, 141
lightning bolts, throwing, 363
mob attack targets, 362-363
mob found somebody to
attack, retrieving, 362
plug-in configuration file, 360
project, creating, 360
source code, 364-366
bigdig command, 295
BigDig mod
block materials, changing, 297
blocks around players,
examining, 296
Bukkit class library,
adding, 294
circles around players,
digging out, 295-297
deploying, 301
generation the completion sound/
message, 297
hole size, determining, 295
plug-in configuration file, 294
project, creating, 294
source code, 298-301
binary value variables, 68
blank spaces (Java programs),
49-50
block statements, 64, 94-96
BlockFace enumeration, 327
blocks, 43
adjacent, retrieving, 327
around players, examining, 296
changing ground under
players. See StoneWalker mod
destroying, 297
generation the Relative() method, 327
generation the Type() method, 250
materials
changing, 297
determining, 250, 310
stone eligibility, 327
tree saplings, planting, 353
right below players,
finding, 327
turning into air, 297
Boole, George, 69
boolean variables, 68-69
brackets
{ } (curly)
arrays, 121
Java programs, 43
[ ] (square), 120, 284
break statement, 98, 113
breaking loops, 113
browse() method, 216
buffered input streams, 229-231
BufferedInputStream()
constructor, 230
Bukkit
Plugins directory website, 368
Project website, 3
Bukkit class library
adding to projects, 245
events
documentation, 346
handling methods, 326
packages, 325
Java documentation, 290, 367
website, 61
Bukkit Minecraft server
Bukkit server TVAR file can’t be
found error, 5
connections
creating, 9-10
fixing, 11-12
downloading, 2
EULA check, 4
Java virtual machine can’t be
found error, 7-9
player profiles, changing, 12
running first time, 4
starting, 3
bytecode, 216
bytes
reading, 215
skipping, 227
streams, 224
variables, 67
writing to streams, 233

B
backing up worlds, 302-304
backslash character (\), 82, 225
carriage return character, 82
bats, summoning with
coordinates, 56
Battlepoint application, 181-183
bedrock, digging, 296
behaviors, 132-135
Benchmark application, 116-117
BestFriendOfZeus mod, 360
Bukkit class library,
adding, 360
deploying, 366
event listener, registering, 362
events, receiving, 362
C
C418, 229
CableModem class, 143
Calculator application, 191-193
cancelling event normal
behaviors, 352
cannot find symbol error
message, 47
carriage return character, 82
case
sensitivity
consistency, 70
variable names, 69
strings, changing, 85
case statements, 97
casting, 136-137
  autoboxing, 141
defined, 137
destinations, 137
objects, 138
sources, 137
unboxing, 141
variables, 137
catching exceptions, 190-191
  handling something after, 196-197
HomePage application, 201-202
multiple exceptions, 194
PageCatalog application, 202-203
throwing exceptions, compared, 190
  try-catch blocks, 191
Calculator application, 191-193
NewCalculator application, 193-196
census
  mobs, 268
    Bukkit class library, adding, 268
  commands, executing, 269
  current mobs, listing, 269
deploying, 278
player messages, sending, 271
plug-in configuration file, 268-269
project creating, 268
source code, 274-277
type counts, storing, 270-273
villagers
  creating, 273-274
deploying, 279
char variables, declaring, 66, 80
characters
  char variables, declaring, 66
defining, 65, 80
quotation marks, 66
special, 81-82
streams, 224
string variables, declaring, 66
  Wheel of Fortune application, 129
checkAuthor() method, 158
CheckDatabase class, 198
chickens
  spawning, 351
  speed, setting, 264
turning into tree laying mutants. See JohnnyApple-chicken mod
zombies riding. See ZombieChicken mod
ChickenStorm mod
  Bukkit class library, adding, 245
chickens, spawning
  babies versus adults, 251
  block materials, determining, 250
  number of, 248
  random locations, 250
  specific locations, 249-250
deploying, 254-255
JAR file, finding, 253
overpopulating, 249
project, creating, 245
source code, 251-253
YAML file, 246-248
chopAdjacentTrees() method, 312-313
chopping down trees. See TreeChopper mod
circle radius, calculating, 295
class statement, 42, 134, 324
classes, 132
  AnimatedLogo, 166
  Applet, 165
  ArrayIndexOutOfBoundsException, 190
  Arrays, 125
  Bukkit, 61
  CableModem, 143
  CheckDatabase, 198
  Console, 230-232
declaring, 42, 212
defining, 25, 57
  Double, 295
  DSLModem, 143
ectrap, 153
event listeners, 321
  Exception, 190
  file, 225
  FileInputStream, 232
  FileOutputStream, 232
  FileReader, 232
  FileWriter, 232
  Font, 186
  helper, 157
  hierarchy, 164-166
HighSpeedModem, 135
  importing, 260-261
inheritance, 135, 164-165
inner, 311
interfaces. See interfaces
ItemSpawnEvent, 352
ItemStack, 281
JApplet, 164-165
libraries, 25-27, 245, 258
LinkRotator, 212
Loc, 311
Logger, 244
Material, 281, 284, 290
Math
documentation, 60
  floor() method, 58
  random() method, 58, 249
  sqrt() method, 53
methods, declaring, 155
Modem, 134, 142
ModemTester, 144
nesting, 157
NetBeans
  automatic importing, 260-261
  compiling, 20
  creating, 18-21
  running, 21
packages, 31
PetWolf program, 31
PlayerInventory, 281
PlayerMoveEvent, 326
Point, 172
Point3D, 172-174
PotionEffect, 264
Properties, 235
RuntimeException, 200
  subclasses. See subclasses
  superclasses, 135
synchronization, 188
System, 230
testing, 174
Thread
  sleep() method, 206
  stop() method, 211
threads, 207, 210-211
variables, 148-150
Vector, 184
Villager, 274
Virus, 148
Clock application, 100-104
ClockTalk application, 104
close() method, 224, 233
closing streams, 224, 233
collections, 171
colors (text output), 340-341
com object, creating, 134-135
Command application
creating, 54
output, 55
source code, 55
commandOn variable, 351
commands
applechicken, 351
bigdig, 295
stonewalk, 322
stopstonewalk, 322
Summon, 56
CommandSender object
verification, 243
comments, 29, 43
Commodity application, 98-99
comparing strings, 84, 92-93
compiling
classes, 20
Java programs, 46
complex for loops, 115
calendar speed, testing, 116-117
concatenating strings, 82-83
concatenation operator (+), 82-83
conditionals, 92
Clock application, 100-104
ClockTalk application, 104
define, 91
if, 92
blocks, 94-96
equal/not equal comparisons, 93
less/greater-than comparisons, 92-93
if-else, 96-97
switch, 97-98
ternary operator (?), 99-100
configuration settings. See
properties
Configurator application, 237-238
ConfigWriter application, 233-235
connections (Minecraft servers)
creating, 9-10
fixing, 11-12
Console application, 230-231
Console class, 230-232
console input, 230
constants, 71
constructors
arguments, 154
BufferedInputStream(), 230
declaring, 154
FileInputStream(), 226
FileOutputStream(), 232
inheritance, 154
Location, 249
PotionEffect(), 264
threads, 213
contains() method, 169
containsKey() method, 186
containsValue() method, 186
continue statement, 113
controlling access. See access control
converting. See also casting properties to numerical values, 236
variables to objects, 139
coordinates (players), finding, 249
counters, 108
createNewFile() method, 225
Creative Commons license, 229
credit card purchase verification application, 198
CreditCardChecker application, 198
Credits application, 87-88
curly brackets { }, 64
arrays, 121
Java programs, 43
current location, retrieving, 29
currentThread() method, 215
damage() method, 347
binary values, 68
boolean, 68-69
byte, 67
cchar, 66
enum, 271
hexadecimal values, 68
integer, 65
long, 67
short, 67
strings, 66, 80
decrement operator (--), 72
decrementing variables, 72-74
default statements, 98
default variables, 149
delete() method, 225
deleting files, 225
deploying mods, 254-255
BestFriendOfZeus, 366
BigDig, 301
HealthChecker, 345
JohnnyApplechicken, 357
MobCensus, 278
PetWolf mod, 34
StoneWalker mod, 331-332
Transmuter mod, 289-290
TreeChopper, 318
ZombieChicken, 264
deprecated, 211
Desktop object, 216
destinations (casting), 137
destroying blocks, 297
dialogs
New Empty Java File, 41
New Project, 39
end-user license agreements (EULAs), 4
endless loops, 112
entities. See packages
EntityDamageEvent object, 338
EntityTargetEvent object, 339
EntityTargetEvent.TargetReason enumeration, 363
entrySet() method, 186, 272
enumerations, 271
BlockFace, 327
EntityTargetEvent.TargetReason, 363
equal sign (=), 66, 70
equality, 93
equals() method, 84, 165
equalsIgnoreCase() method, 243
errors
arrays, 122
defined, 189
exceptions, 190-191
ArrayIndexOutOfBoundsException, 190
catching multiple, 194-196
catching with try-catch blocks, 191-194
creating, 199
defined, 189
Exception class, 190
handling something after, 196-197
HomePage application, 201-202
ignoring, 199-200
PageCatalog application, 202-203
throwing, 197-199
throwing versus catching, 190
unchecked, 200-201
handling, 194, 213-214
IndexOutOfBoundsException, 200
Java programs, 47-48
MalformedURLException, 199
Minecraft server
Bukkit server JAR file can’t be found error, 5
Java virtual machine can’t be found error, 7-9
NetBeans, fixing, 22-23, 259
NullPointerException, 200
NumberFormatException, 194, 197
escape sequences, 81
EULAs (end-user license agreements), 4
@EventHandler annotation, 326
events
defined, 321
documentation, 346
explosions, 346
handling, 321
adjacent blocks, retrieving, 327
block materials, transforming, 328
blocks right below players, finding, 327
defined, 324
@EventHandler annotation, 326
graphical user interfaces, 324
methods, 326
mod on status, checking, 327
player locations, 326-327
registerEvents() method, 325
lightning bolts, throwing, 363
listening, 321
mob current/maximum health, determining, 339
mobs taking damage, 338
mobs targeting players for attack, 339
mods as listeners, registering, 324-325, 362
StoneWalker mod, 324
mobs
attack target reasons, 363
found somebody to attack, retrieving, 362
targets, identifying, 362
mods, receiving, 362
mouse clicks, handling, 216
normal behavior, cancelling, 352
packages, 325
player caused, 326
exclamation points (mob health), calculating, 340
Exception class, 190
exceptions, 122, 190
array index out of bounds, 122, 190
catching, 190-191
  multiple, 194-196
  try-catch blocks, 191-194
  versus throwing, 190
creating, 199
defined, 189
Exception class, 190
handling something after, 196-197
HomePage application, 201-202
ignoring, 199-200
PageCatalog application, 202-203
throwing, 197-199
unchecked, 200-201
executeCommand() method, 243, 284-285
exists() method, 225
exiting loops, 113
expressions, 75. See also
  operators
  advantages, 76
  defined, 64
  increment/decrement operators, 73
  operator precedence, 74-75
PlanetWeight application, 76-78
extends statement, 143, 165
File class, 225
FileInputStream class, 232
FileInputStream() constructor, 226
FileOutputStream class, 232
FileOutputStream() constructor, 232
FileReader class, 232
files
  creating, 225
deleting, 225
disk existence, checking, 225
JAR. See JARs
names, 225
properties, 235
Configurator application, 237-238
converting to numerical values, 236
displaying all, 236
loading, 236
names, 235
retrieving, 236
storing, 236
reading, 226
FileWriter class, 232
finding
  blocks right below players, 327
  mods, 368
  outside-the-cube trees, 311-313
  player coordinates, 249
strings, 86
FindPrimes application, 208-211
floated statement, 65
float variable, 65
floating-point numbers
  declaring, 65
  rounding down, 58
floor() method, 58
folders
  contents, listing, 226
  creating, 225
Font class, 186
FontMapper application, 186-187
Football application, 94-95
for loops, 108-109
array lists, 170
complex, 115
counter variables, 108
empty sections, 115
exiting, 113
Nines application, 109-110
spawning chickens, 351
syntax, 108-109
ifstreams character, 82
forward slashes (/), 225
framework (mods), 242-245
CommandSender object
  verification, 243
equalsIgnoreCase() method, 243
executeCommand() method, 243
onCommand() method, 243
player locations, 244
Player object, creating, 244
server messages, logging, 244
World object access, 244
friends, creating. See BestFriendOfZeus mod
Game application, 94-95
gameOver variable, 69
Gauntlet, 49
generics, 178
get() method
  array lists, 169
  hash maps, 185
getAmount() method, 286
getBlock() method, 250, 327
getBytes() method, 233
getFrom() method, 327
getHealth() method, 339
getIdentifier() method, 153
getInventory() method, 285
getItem() method, 286
gKey() method, 272
gLivingEntities() method, 269
gLocation() method, 29, 244
gMaterial() method, 285
gMaxHealth() method, 339
gMaxStackSize() method, 286
getName() method, 225
gOrDefault() method, 185
gPlayer() method, 326
gPluginManager() method, 325
gProperty() method, 236
gReason() method, 363
gRelative() method, 327
gSize() method, 285
gTarget() method, 362
JARS (JAVA ARCHIVE FILES) 379

getTo() method, 327
gETType() method, 250, 272
gETValue() method, 214
gETvirusCount() method, 160
gETWorld() method, 29, 244
GitHub mod source code, 368
godfather, 304
graphical user interfaces, 324
greater-than conditional, 92-93
greater than operator, 92
greeting variable, 44
ground under players, changing. See StoneWalker mod
grouping statements. See blocks

h

handling events. See events
hash maps, 184
capacity, 184
creating, 184
FontMapper application, 186-187
keys/values presence, determining, 186
load factors, 184
looping through, 186
mobs, 270-273
objects, 185
phonebook, 185
size, 186
health bars, 339-340
healthcheck on/healthcheck off commands, 338
HealthChecker mod, 336
Bukkit class library, adding, 336
current maximum health, determining, 339
deploying, 345
display time, 345
health bars
colors, selecting, 340-341
creating, 339-341
current health points, calculating, 340
displaying, 340
scale, 340
mobs
taking damage, 338
targeting players for attack, 339
plug-in configuration file, 337

How can we make this index more useful? Email us at indexes@quepublishing.com

byte, 67
declaring, 65
floating-point, 58
hexadecimal variables, 68
identifier variable, 148
long, 67
random, creating, 58
short, 67
underscores, 67
integrated development environments (IDEs), 15. See also NetBeans
interfaces, 136
ActionListener, 212
Listener, 324
Runnable, 206-207
inventories
loading, 285
looping through, 286
materials. See materials
slots, determining, 285
I/O (input/output), 223
console input, 230
files. See files
streams
buffered input, 229-231
bytes, 224
close, 224
closing, 224, 233
define, 224
reading data from, 226-229
types, 224
data writing to, 232-235
isAdult() method, 274
isStoneWalking variable, 323, 327
item stacks, 285-286
ItemSpawnEvent class, 352
ItemSpawnEvent object, 352
ItemStack class, 281
iteration. See loops
iterators, 109

J

JApplet class
inheritance, 164-165
methods, 165
JARs (Java archive files), 2
ChickenStorm, 253
PetWolf, finding, 33
ZombieChicken, 264

ID3Reader application, 227-229
IDE (integrated development environment), 15. See also NetBeans
identifier variable, 148
if statements, 92
blocks, 94-96
equal/not equal comparisons, 93
less/greater-than comparisons, 92-93
if-else statements, 96-97
ignoring exceptions, 199-200
implements keyword
class statement, 324
Runnable interface, 207
importing classes, 260-261
incrementing variables, 72-74
increment operator (++), 72
indexOf() method, 86-87, 180
IndexOutOfBoundsException error, 200
inequality operator (!=), 94
infinite loops, 112
inheritance, 135, 163-165
constructors, 154
hierarchy, 135
JApplet class, 164-165
initializing counter variables, 108
inner classes, 311
input/output. See I/O
installing NetBeans, 16
instance variables, 134
int statement, 65
integers
arrays
declaring, 120
Wheel of Fortune application, 129
binary values, 68
Java
archive files. See JARs
Bukkit class library
documentation, 290
Class Library, 56
classes, 57
documentation, 57, 61
packages, 57
Development Kit (JDK), 8
Material class
documentation, 290

Java

keys (hash maps), 186
keywords
implements
class statement, 324
Runnable interface, 207
this, 325

length() method, 85, 225
length variable, 122-124
less than operator (<), 92-93
letter frequency application,
127-130
libraries
adding to projects, 27
Apache Project, 57
Bukkit. See Bukkit class library
class, 25
Java class, 56-57, 61
lightning bolts, throwing, 363
LinkRotator application
class declaration, 212
constructors, 213
error handling, 213-214
mouse clicks, handling, 216
revolving link, displaying,
217-219
threads
running, 215-216
starting, 214
stopping, 220
variables, setting up,
212-213
LinkRotator class, 212
links (revolving), displaying,
217-219
list() method, 236
Listener interface, 324
listening (events), 321
mobs
current/maximum health,
determining, 339
taking damage, 338
targeting players for attack,
339
mods as listeners,
registering, 324-325, 362
player locations, 326-327
StoneWalker mod, 324
listFiles() method, 226

listings
Battlepoint application,
181-183
Benchmark application,
116-117
BestFriendOfZeus.java,
364-366
BigDig.java, 298-301
CableMode class, 143
Calculator application,
191-192
ChickenStorm.java, 251-253
Clock application, 100-103
Command application source
code, 55
Commodity application, 98-99
Configurator application,
237-238
ConfigWriter application,
233-234
Console application, 230-231
Credits application, 87
Dice application, 59
DslModem class, 143
FindPrimes application,
208-210
FontMapper application,
186-187
Game application, 94-95
HealthChecker.java,
342-345
HomePage application,
201-202
ID3Reader application,
227-229
JohnnyApplechicken.java,
354-357
Minecraft servers, starting, 3
MobCensus.java, 274-277
Modem class, 142
ModemTester class, 144
NameSorter application, 125
NewCalculator application,
193-194
NewRoot application, 140
Nines application, 109-110
NumberDivider application,
195-196
PageCatalog application,
202-203
PetWolf.java
full text, 29-31
plugin.yml, 31-32
PlanetWeight application,
76-77

Runtime Environment (JRE), 8
java.io package, 58, 223
java.time package, 58
java.util package, 58, 235
JDK (Java Development Kit), 8
JohnnyApplechicken mod
Bukkit class library,
adding, 350
cancelling chickens normal egg
laying behavior, 352
deploying, 357
event handler,
registering, 352
player location, holding, 351
plug-in configuration file, 350
project, creating, 350
source code, 354-357
spawning chickens, 351
tree laying capabilities,
turning on/off, 351
tree sapling supported
blocks, 353
JRE (Java Runtime
Environment), 8

K

L
METHODS

plugin.yml, 247-248
BestFriendOfZeus mod, 361
BigDig mod, 294
HealthChecker mod, 337
JohnnyApplechicken mod, 350
MobCensus mod, 268
StoneWalker mod, 322
Transmuter mod, 283
TreeChopper mod, 309
ZombieChicken mod, 259
Point3D class, 172-173
PointTester application, 174-175
Root application source code, 52-53
SpaceRemover application, 127
Spartacus class, 20-21
Splash program, 41, 45
StoneWalker.java, 328-331
StringLister application, 170-171
Transmuter.java, 287-289
TreeChopper.java, 314-317
Variable application, 66
Viru application, 159
VirusLab application, 160-161
Wheel of Fortune application, 127
ZombieChicken.java, 261-264
literals, 70
load factors (hash maps), 184
loading properties, 236
Loc class, 311
locations, 29, 132
block materials, determining, 250
constructor, 249
getBlock() method, 250
mobs
block materials, determining, 250
distances, calculating, 273
distances, retrieving, 273
specific, 249-250
players
distances, calculating, 273
holding, 351
listening, 326-327
searched, storing, 313
spot variable, 351
LOG variable, 244
Logger class, 244
logging server messages, 244
long variables, 67
loops
array lists, 170
Benchmark application, 116-117
blocks around players, examining, 296
defined, 107
do-while, 112-113
executing, 113
for, 108-109
array lists, 170
complex, 115
counter variables, 108
determining, 250
empty sections, 115
Nines application, 109-110
swamping chickens, 351
syntax, 108-109
hash maps, 186
infinite loops, 112
mob census, 272
names, 114-115
nested, 114, 130
player inventories, 286
TreeChopper mod, 312
while, 111
lowercase. See case

M

main statement, 42
makeBarGraph() method, 340
MalformedURLException errors, 199
maps (hash), 184
map capacity, 184
creating, 184
FontMapper application, 186-187
keys/values presence, determining, 186
load factors, 184
looping through, 186
phonebook, 185
retrieving objects, 185
size, 186
storing objects, 185
Material class, 281, 284, 290
materials
blocks
changing, 297
determining, 250, 310
tree saplings, planting, 353
names, 284
transmuting. See Transmuter mod
turning to stone eligibility, 327
Math class
documentation, 60
floor() method, 58
random() method, 58, 249
sqrt() method, 53
memory
controlling, 162
server lags, 310
messages
colors, selecting, 340-341
digging holes
completion, 298
TreeChopper mod, 314
methods
accessor, 153
actionPerformed(), 216
add(), 165, 169, 178
addPotionEffect(), 264
alert(), 153
arguments, 153
browse(), 216
checkAuthor(), 158
chopAdjacentTrees(), 312-313
classes, declaring, 155
close(), 224, 233
constructors, 154
contains(), 169
containsKey(), 186
containsValue(), 186
createNewFile(), 225
currentThread(), 215
damage(), 347
declaring, 151
defined, 29, 85, 147, 151
delete(), 225
displaySpeed(), 134
determineSet(), 186, 272
equals(), 84, 165
equalsIgnoreCase(), 243
events
handling, 326
learning, 346
executeCommand(), 243, 284-285
exists(), 225
floor(), 58
get()
array lists, 169
hash maps, 185
getAmount(), 286
getBlock(), 250, 327

How can we make this index more useful? Email us at indexes@quepublishing.com
getBytes(), 233
getEntity(), 362, 338
getFrom(), 327
getHealth(), 339
getInventory(), 285
getItem(), 286
getKey(), 272
getLivingEntities(), 369
getLocation(), 29, 244
getMaterial(), 285
getMaxHealth(), 339
getMaxStackSize(), 286
getName(), 225
getOrDefault(), 185
getPlayer(), 326
getPluginManager(), 325
getProperty(), 236
getReason(), 363
getRelative(), 327
getSeconds(), 153
getSize(), 285
getTarget(), 362
getTo(), 327
type(), 250, 272
getURI(), 214
getValue(), 273
getVirusCount(), 160
getWorld(), 29, 244
indexOf(), 86-87, 180
isAdult(), 274
length(), 85, 225
list(), 236
listFiles(), 226
makeBarGraph(), 340
mod on status, checking, 327
okToTransform(), 328
onCommand(), 329
arguments, 243
StoneWalker mod, 324
onEnable(), 362
onEntityDamage(), 338
onEntityTarget(), 339, 362
onItemSpawn(), 352
onPlayerMove(), 326
overriding, 165-166, 324
paint(), 166
parseDouble(), 295
parseInt(), 139, 155
playSound(), 297
println(), 77, 80-81, 151
public, 152
put(), 185
random(), 58, 249
read(), 226
readLine(), 232
recursion, 313
registerEvents(), 325
generateTo(), 225
return values, 152
run(), 207, 215-216
sendMessage(), 271
setAmount(), 286
setBaby(), 263
setBackground(), 165
setCustomName(), 340
setIdentifier(), 153
setLayout(), 165
setPassenger(), 263
setProperty(), 236
setType(), 286
setYield(), 346
shoot(), 183
showHealth(), 339
showVirusCount(), 155
signatures, 153
size()
array lists, 179
hash maps, 186
skip(), 227
sort(), 125
spawn(), 29, 250
sqrt(), 53
start(), 214
stop(), 211
store(), 236
strikeLightning(), 363
substring(), 229
toCharArray(), 122
toLowerCase(), 85
toUpperCase(), 85
variable scope, 155-157
write(), 232
Minecraft
EULA website, 4
Mods Programming Absolute
Beginner’s Guide
website, 371
projects
creating, 39
opening, 40
servers, 2
Bukkit API and server
downloads, 2
Bukkit server JAR file can’t
be found error, 5
connections, 9-12
EULA check, 4
Java virtual machine can’t
be found error, 7-9
messages, logging, 244
player profiles, changing, 12
running first time, 4
starting, 3
website, 9
minus sign (-), 72
MobCensus mod, 268
Bukkit class library,
adding, 268
commands, executing, 269
current mobs, listing, 269
deploying, 278
player messages, sending, 271
plug-in configuration file,
268-269
project, creating, 268
source code, 274-277
type counts, storing, 270-273
villagers, 273-274, 279
mobs
adult/child forms, 263
census, 268
Bukkit class library,
adding, 268
commands, executing, 269
current mobs, listing, 269
deploying, 278
player messages, sending, 271
plug-in configuration file,
268-269
project, creating, 268
source code, 274-277
type counts, storing,
270-273
current health, calculating,
339-340
defined, 26
found somebody to attack, retrieving, 362
hash map, 270-273
health state during battle,
displaying. See
HealthChecker mod
lightning bolts, throwing at, 363
locations
block materials, determining, 250
random, 250
retrieving, 273
specific, 249-250
numbers, 248
overpopulating, 249
passengers, adding, 263
player distance from,
calculating, 273
potions, adding, 263-264
taking damage, 338
targeting players for
attack, 339
targets, 362-363
types, 270
wolf, creating, 29
classes, 31
deploying, 34
full text, 29-32
JAR file, finding, 33
troubleshooting, 34
Modem class, 134, 142
modems, 133
CableModem class, 143
DslModem class, 143
Modem class, 142
ModemTester class, 144
ModemTester class, 144
mods. See also specific names of mods
as event listeners, registering, 324-325, 362
defined, 26
finding, 368
framework, 242-245
CommandSender object verification, 243
equalsIgnoreCase() method, 243
executeCommand() method, 243
onCommand() method, 243
Player locations, retrieving, 244
Player object, creating, 244
server messages, logging, 244
World object access, 244
on status, checking, 327
player information, 29
program, creating, 27-28
projects, 26-27
source code, displaying on GitHub, 368
storing, 26
turning on/off, 323
modulus operator (%), 72
Monitor objects, 133
mouse clicks, handling, 216
MP3 files, reading, 227-229
multidimensional arrays, 124-125
multiplication operator (*), 72
multithreading, 206

N
names
  event handling methods, 326
  loops, 114-115
  materials, 284
  packages, 58
  properties, 235, 247
  variables, 69-70
  worlds, 303
NameSorter application, 125-126
nesting
  classes, 157
  loops, 114, 130
NetBeans
  automatic class importing, 260-261
  classes
    compiling, 20
    creating, 18-21
    running, 21
  errors, fixing, 22-23, 259
  installing, 16
  overview, 15
  projects, creating, 17-18
  source editor, 46
  user interface, 16
  website, 23
NewRoot application, 140-141
New Empty Java File dialog, 41
New File Wizard, 40
New Project dialog, 39
New Project Wizard, 17
NewCalculator application, 193-194
newline characters, 82
new statement, 120, 154
Nines application, 109-110
NullPointerException error, 200
NUM_CHICKENS variable, 248
NumberDivider application, 195-196
NumberFormatException errors, 194, 197
numbers
  floating-point, 65
  integers, declaring, 65
  prime sequence, displaying, 208-211

O
object-oriented programming. See OOP
objects
  advantages, 133
  attributes, 132, 147
  behavior, 132
  casting, 138
  classes, 132
  converting to variables, 139
  creating, 133-135, 142-144
  CableModem class, 143
  DslModem class, 143
  Modem class, 142
  ModemTester class, 144
  debugging, 133
  inheritance, 163-165
  hierarchy, 135
  JApplet class, 165
  instance variables, 134
  referencing, 158
  Server, creating, 325
  sharing, 168
  storing, 168-170
  thisThread, 215
  unboxing, 141
  variables, 148-149
okToTransform() method, 328
onCommand() method, 269
arguments, 243
StoneWalker mod, 324
onEnable() method, 362
onEntityDamage() method, 338
onEntityTarget() method, 339, 362
onItemSpawn() method, 352
onPlayerMove() method, 326
OOP (object-oriented programming), 131-132
advantages, 133
debugging, 133
encapsulation, 153
inheritance, 135, 163-165
  hierarchy, 135
  JApplet class, 165
objects. See objects
operators
  +=, 84
  addition (+), 72
  concatenation (+), 82-83
  decrement (--), 72
  division (/), 72
  equality (=), 93
greater than (>), 92
increment (++;), 72
inequality (!=), 94
less than (<), 92
modulus (%), 72
multiplication (*), 72
precedence, 74-75
prefixing/postfixing, 73-74
subtraction (-), 72
ternary (?), 72

other operators

order of precedence, 74-75
organizing applications, 94-96
output. See I/O (input/output)
outside-the-cube trees, 310-313
overpopulating with mobs, 249
@Override annotation, 324
overriding methods, 165-166, 324

P

packages, 149
defined, 31, 57
events, 325
java.io, 58, 223
java.time, 58
java.util, 58, 235
names, 58
PageCatalog application, 202-203
pageLink array, 213
pageTitle array, 212-213
paint() method, 166
parseDouble() method, 295
parseInt() method, 139, 155
passengers (mobs), adding, 263
passing arguments
applications, 54
methods, 153
pasting strings, 82-83
percent sign (%), 72
PetWolf mod
classes, 31
deploying, 34
full text, 29-31
JAR file, finding, 33
Java class libraries, adding, 27
player information, 29
plugin, creating, 31-32
program, creating, 27-28
project, 26
troubleshooting, 34
Wolf object, 29
phonebook hash map, 185-186
PlanetWeight application, 76-78
planting tree saplings, 353
Player objects, 29, 244
PlayerInventory class, 281
PlayerInventory object, 285
PlayerMoveEvent class, 326
players
40 block cube around, 309
blocks right below, finding, 327
coordinates, finding, 249
events, causing, 326
ground under, changing. See StoneWalker mod
information, 29
inventories
block materials,
changing, 297
block materials,
determining, 310
blocks into air,
changing, 297
loading, 285
looping through, 286
material names, 284
material transmutations.
See Transmuter mod
slots, determining, 285
locations
holding, 351
listening, 326-327
retrieving, 244
mob distance from,
calculating, 273
profiles, changing, 12
targeting for attack, 339
world, accessing, 244
playing sounds, 297
playSound() method, 297
Plugin Manager object, 325
plugin.yml files
BestFriendOfZeus mod, 360
BigDig mod, 294
text listing, 247-248
HealthChecker, 337
JohnnyApplechicken mod, 350
MobCensus mod, 268-269
PetWolf mod, 31-32
StoneWalker mod, 322
storing, 248
Transmuter mod, 283-284
TreeChopper mod, 308
ZombieChicken mod, 258-259
plus sign (+)
addition operator, 72
catatonation operator, 82-83
increment operator (+;), 72
Point class, 172
Point3D class, 172-174
points, creating, 180-183
PointTester application, 174-175
postfixing, 73-74
PotionEffect class, 264
PotionEffect() constructor, 264
potions, adding to mobs, 263-264
prefixing, 73-74
prime numbers sequence, displaying, 208-211
printing strings, 80
println() method, 80-81
special characters, 81-82
println() method, 77, 80-81, 151
private variables, 149
programs (Java)
adding to open projects, 40
blank spaces/whitespace, 49-50
blocks, 43
{} (curly brackets), 43
cannot find symbol error, 47
category, choosing, 41
class statement, 42
comments, 29, 43
compiling, 46
creating, 27-28
defined, 38
debugging, 46
line numbers for errors, 48
main statement, 42
running, 48
saving, 45
source code, 41
spelling/punctuation
erors, 48
variables, 44
projects
BestFriendOfZeus mod, 360
BigDig, 294
ChickenStorm, creating, 245
creating, 26
HealthChecker mod, 336
Java class libraries, adding, 27
JohnnyApplechicken, 350
Minecraft, 39-40
MobCensus, 268
NetBeans, 17-18
StoneWalker, 322
Transmuter mod, 283
TreeChopper, 308
ZombieChicken, 258
properties, 235
  Configurator application, 237-238
  converting to numerical values, 236
  displaying all, 236
  loading, 236
  names, 235, 247
  retrieving, 236
  storing, 236
  Properties class, 235
  protected variables, 149
  public methods, 152
  public statements, 134
  public variables, 149
  put() method, 185
  Pythagorean theorem, 273

Q

question mark (?), 99-100
  quotation marks
    double ("), 66, 81
    escape codes, 82
    single (‘), 66

R

radius, calculating, 295
  random() method, 58, 249
  random numbers, 58
  read() method, 226
  reading
    buffered input streams, 230
    files, 226
    ID3Reader application, 227-229
    multiple bytes, 227
    single bytes, 226
    skipping bytes, 227
    readLine() method, 232
  recursion, 313
  redwoods, 310
  referencing objects, 158
  registerEvents() method, 325
  registering mods as event listeners, 324-325, 362
  removing objects from arrays, 170, 179
  renameTo() method, 225
  renaming files, 225
  restricting access. See access control

return values (methods), 152
  revolving links, displaying, 217-219
  Rock Proper, 229
  Root application, 52-53
  Rosenfeld, Daniel, 229
  run() method, 207, 215-216
  Runnable interface, 206-207
  running
    applications, 21
    Java programs, 48
    threads, 215-216
  RuntimeException class, 200

S

saving
  Java programs, 45
  properties, 236
  worlds, 302-304
  scope (variables), 155-157
  Seed numbers, 303
  semicolons (;), 44, 115
  sendMessage() method, 271
  servers
    lag, 310
    messages, 314
    objects, creating, 325
  setAmount() method, 286
  setBaby() method, 263
  setBackground() method, 165
  setCustomName() method, 340
  setIdentifier() method, 153
  setLayout() method, 165
  setPassenger() method, 263
  setType() method, 286
  setYield() method, 346
  sharing objects, 168
  shoot() method, 183
  shooting targets, creating, 181-183
  short variables, 67
  showHealth() method, 339
  showVirusCount() method, 155
  signatures (methods), 153
  single quotation marks (‘), 66, 82
  size
    array lists, 179
    hash maps, 186
    holes, determining, 295
  size() method
    array lists, 179
    hash maps, 186
    slashes (‘/’), 43
    sleep() method, 206
    slowing down applications, 206
    sort() method, 125
    sorting arrays, 125-126
    sounds, playing, 297
    source code editors, 38
    source editor (NetBeans), 46
    sources (casting), 137
    SpaceRemover application, 123-124
    spacing (Java programs), 49-50
    Spartacus application, 20-23
    errors, fixing, 22-23
    running, 21
    source code, 20-21
    spawn() method, 29, 250
    spawn points, 319
    spawning
      chickens, 351
      mobs
        block materials, 250
        determining, 250
        numbers, 248
        random locations, 250
        specific locations, 249-250
      special characters, 81-82
      splash messages website, 49
    Splash program
      adding to open projects, 40
      blank spaces/whitespace, 49-50
      blocks, 43
      { (curly brackets), 43
      category, choosing, 41
      class statement, 42
      comments, 43
      compiling, 46
      errors
        cannot find symbol, 47
        displaying, 46
        line numbers, 48
        spelling/punctuation, 48
        line-by-line breakdown, 45
      main statement, 42
      running, 48
      saving, 45
      source code, 41
      variables, 44
      spot variable, 351

How can we make this index more useful? Email us at indexes@quepublishing.com
creating, 207, 210-211
error handling, 213-214
mouse clicks, handling, 216
reversing links, displaying, 217-219
Runnable interface, 206
running, 215-216
slowing down applications, 206
starting, 214
stopping, 220
Thread class, 206
variables, setting up, 212-213
throw statements, 197
throwing exceptions, 197-199
catching exceptions, compared, 190
HomePage application, 201-202
PageCatalog application, 202-203
throwing lightning bolts, 363
ticks, 264
timekeeping, 100-104
toCharArray() method, 122
toLowerCase() method, 85
tops variable, 65
toUpperCase() method, 85
transforming block materials, 328
Transmuter mod, 282
Bukkit class library, adding, 283
deploying, 289-290
input/output materials, defining, 284-285
player inventory loading, 285
looping through, 286
slots, determining, 285
plug-in configuration file, 283-284
project, creating, 283
source code, 287-289
TreeAssist mod, 368
TreeChopper mod, 308
40 block cube around players, 309
block material, determining, 310
Bukkit class library, adding, 308
deploying, 318
outside-the-cube trees, 310-313
plug-in configuration file, 308
project, creating, 308
server message, 314
source code, 314-317
trees
chopping down. See TreeChopper mod
creating with mutated chickens. See JohnnyApple-chicken mod
outside-the-cube, 310-313
redwoods, 310
saplings, planting, 353
website, 311
troubleshooting
Minecraft servers
Bukkit server JAR file can’t be found error, 5
connection problems, 11-12
Java virtual machine can’t be found error, 7-9
PetWolf mod, 34
try-catch blocks (exceptions, catching), 191
Calculator application, 191-193
multiple classes, 194-196
NewCalculator application, 193-194
try-catch-finally blocks, 196-197
try-catch statements, 214
turning blocks into air, 297
turning on/off mods, 323
unboxing, 141
unchecked exceptions, 200-201
underscores (_), large numbers, 67
variable names, 69
upper limits (arrays), 122
uppercase. See case
URISyntaxException errors, 214
user interfaces (NetBeans), 16
values
class variables, changing, 150
hash maps, 186
return, 152
starting, 70-71
variables
incrementing/decrementing, 72-74
storing, 70-71
Variable application, 65-66
variables, 44
=(equal signs), 66
access control, 148-149
arrays
character, 122
declaring, 120
defined, 119
elements, 120-121
exceptions, 122
initial values, 120
integers, 120, 129
multidimensional, 124
sample application, 123-124
sorting, 125-126
strings, 120
upper limits, checking, 122
Wheel of Fortune application, 127-130
casting, 137
class, 150
cmdOn, 351
constants, 71
contents, displaying, 44
converting to objects, 139
counter, 108
creating, 64-65
data types, 44
declaring, 148-149
default, 149
defined, 63
display, 338
identifier, 148
instance, 134
isStoneWalking, 323, 327
length, 122-124
LinkRotator application, 212-213
LOG, 244
names, 69-70
NUM_CHICKENS, 248
private, 149
protected, 149
quotation marks, 66
referencing, 158
scope, 155-157
spot, 351
strings
case, changing, 85
comparing, 84
concatenating, 82-83
How can we make this index more useful? Email us at indexes@quepublishing.com
Credits application, 87-88
declaring, 80
displaying, 80-81
finding, 86
length, determining, 85
linking, 83-84
special characters, 81-82
types
  - binary values, 68
  - boolean, 68-69
  - byte, 67
  - char, 66, 80
  - floating-point, 65
  - hexadecimal values, 68
  - integers, 65
  - long, 67
  - short, 67
  - string, 66
underscores (_), 67
values
  - incrementing/decrementing, 72-74
  - starting, 70-71
  - storing, 70-71
warning, 156
Vector class, 184
vectors, 179
Villager class, 274
villagers census
  - creating, 273-274
  - deploying, 279
Virus application
  - class constructor, 154
  - methods, 153
  - showVirusCount() method, 155
source code, 159
Virus class, 148
VirusLab application, 160-161
void statement, 152
Volume Alpha (Rosenfeld), 229
Volume Beta (Rosenfeld), 229
Creative Commons license, 229
events documentation, 346
Java Class Library documentation, 57
Minecraft, 9
  - EULA, 4
  - splash messages, 49
NetBeans, 16, 23
Rock Proper, 229
TreeAssist mod directory page, 368
trees, 311
Web-Adventures, 232
Wheel of Fortune application,
  - 127-130
Wheel of Fortune, 127
while loops, 111-113
whitespace (Java programs),
  - 49-50
wizards
  - New File, 40
  - New Project, 17
Wolf object, 29
wolves, creating, 29
classes, 31
deploying, 34
full text, 29-31
JAR file, finding, 33
plugin, creating, 31-32
troubleshooting, 34
Worlds, 29, 244
godfather, 304
names, 303
saving, 302-304
seed numbers, 303
write() method, 232
writing
  - files, 232
  - bytes, 233
  - ConfigWriter application,
    - 233-235
  - output stream, creating, 232
  - write() method, 232
ZombieChicken mod
  - adding, 258
class library, 258
  - adding, 258
chicken speed, setting, 264
deploying, 264
full source code, 261-264
NetBeans automatic class
  - importing, 260-261
plug-in configuration file, cre-ating, 258-259
project, creating, 258
zombies
  - adult/child forms, 263
  - flesh, transmuting into diamonds, 282
  - mounted on chickens. See ZombieChicken mod
pigments, summoning with coordinates, 56