

Rogers Cadenhead

EIGHTH  
EDITION

Covers Java 9  
and Android

Sams **Teach Yourself**

# Java™

in **24**  
**Hours**

SAMS

Rogers Cadenhead

Sams **Teach Yourself**

# Java

in **24**  
**Hours**

Eighth Edition



**Sams Teach Yourself Java in 24 Hours, Eighth Edition**

Copyright © 2018 by Pearson Education, Inc.

ISBN-13: 978-0-672-33794-9

ISBN-10: 0-672-33794-0

Library of Congress Control Number: 2017946911

Printed in the United States of America

# Contents at a Glance

## **PART I GETTING STARTED**

<b>1</b>	Becoming a Programmer .....	3
<b>2</b>	Writing Your First Program .....	13
<b>3</b>	Vacationing in Java .....	27
<b>4</b>	Understanding How Java Programs Work .....	41

## **PART II LEARNING THE BASICS OF PROGRAMMING**

<b>5</b>	Storing and Changing Information in a Program .....	53
<b>6</b>	Using Strings to Communicate .....	71
<b>7</b>	Using Conditional Tests to Make Decisions .....	83
<b>8</b>	Repeating an Action with Loops .....	99

## **PART III WORKING WITH INFORMATION IN NEW WAYS**

<b>9</b>	Storing Information with Arrays .....	111
<b>10</b>	Creating Your First Object .....	125
<b>11</b>	Describing What Your Object Is Like .....	141
<b>12</b>	Making the Most of Existing Objects .....	159

## **PART IV MOVING INTO ADVANCED TOPICS**

<b>13</b>	Storing Objects in Data Structures .....	173
<b>14</b>	Handling Errors in a Program .....	187
<b>15</b>	Creating a Threaded Program .....	203
<b>16</b>	Using Inner Classes and Closures .....	221

## **PART V PROGRAMMING A GRAPHICAL USER INTERFACE**

<b>17</b>	Building a Simple User Interface .....	237
<b>18</b>	Laying Out a User Interface .....	259
<b>19</b>	Responding to User Input .....	273

## **PART VI WRITING INTERNET APPLICATIONS**

<b>20</b>	Reading and Writing Files .....	291
<b>21</b>	Using Java 9's New HTTP Client .....	309
<b>22</b>	Creating Java2D Graphics .....	325
<b>23</b>	Creating Minecraft Mods with Java .....	341
<b>24</b>	Writing Android Apps .....	375

## **APPENDIXES**

<b>A</b>	Using the NetBeans Integrated Development Environment .....	399
<b>B</b>	Where to Go from Here: Java Resources .....	407
<b>C</b>	This Book's Website .....	411
<b>D</b>	Fixing a Problem with the Android Studio Emulator .....	413
<b>E</b>	Fixing Package Not Visible Errors in NetBeans .....	419

# Contents

Introduction	1	<b>HOURL 4: Understanding How Java Programs Work</b>	<b>41</b>
<b>PART I: Getting Started</b>		Creating an Application	41
<b>HOURL 1: Becoming a Programmer</b>	<b>3</b>	Sending Arguments to Applications	43
Choosing a Language	4	The Java Class Library	45
Telling the Computer What to Do	6	Trying Java Statements in JShell	49
How Programs Work	7	<b>PART II: Learning the Basics of Programming</b>	
When Programs Don't Work	8	<b>HOURL 5: Storing and Changing Information in a Program</b>	<b>53</b>
Choosing a Java Programming Tool	9	Statements and Expressions	53
Installing a Java Development Tool	10	Assigning Variable Types	54
<b>HOURL 2: Writing Your First Program</b>	<b>13</b>	Integers and Floating-Point Numbers	54
What You Need to Write Programs	13	Characters and Strings	55
Creating the <code>Saluton</code> Program	14	Other Numeric Variable Types	56
Beginning the Program	14	The <code>boolean</code> Variable Type	57
The <code>class</code> Statement	16	Naming Your Variables	58
What the <code>main</code> Statement Does	17	Storing Information in Variables	59
Those Squiggly Bracket Marks	17	All About Operators	60
Storing Information in a Variable	18	Incrementing and Decrementing a Variable	61
Displaying the Contents of a Variable	18	Operator Precedence	63
Saving the Finished Product	19	Using Expressions	64
Compiling the Program into a Class File	20	<b>HOURL 6: Using Strings to Communicate</b>	<b>71</b>
Fixing Errors	20	Storing Text in Strings	71
Running a Java Program	22	Displaying Strings in Programs	72
<b>HOURL 3: Vacationing in Java</b>	<b>27</b>	Using Special Characters in Strings	73
First Stop: Oracle	27	Pasting Strings Together	74
A Brief History of Java	29	Using Other Variables with Strings	74
Going to School with Java	30	Advanced String Handling	75
Lunch at Food Network	32	Comparing Two Strings	76
Watching the Skies at NASA	33	Determining the Length of a String	77
Getting Down to Business	34		
Stopping by SourceForge for Directions	35		

Copying a String with a New Case	77	Building an Inheritance Hierarchy	130
Looking for a String	77	Converting Objects and Simple Variables	131
Presenting Credits	78	Casting Simple Variables	131
		Casting Objects	132
		Converting Simple Variables to Objects and Back	133
<b>HOURL 7: Using Conditional Tests to     Make Decisions</b>	<b>83</b>	Creating an Object	135
if Statements	84		
Less-Than and Greater-Than Comparisons	84	<b>HOURL 11: Describing What Your     Object Is Like</b>	<b>141</b>
Equal and Not Equal Comparisons	85	Creating Variables	141
Organizing a Program with Block Statements	85	Creating Class Variables	143
if-else Statements	87	Creating Behavior with Methods	144
switch Statements	87	Declaring a Method	145
The Ternary Operator	90	Similar Methods with Different Arguments	146
Watching the Clock	91	Constructors	147
		Class Methods	148
<b>HOURL 8: Repeating an Action with Loops</b>	<b>99</b>	Variable Scope within Methods	148
for Loops	99	Putting One Class Inside Another	149
while Loops	102	Using the this Keyword	151
do-while Loops	103	Using Class Methods and Variables	152
Exiting a Loop	104		
Naming a Loop	105	<b>HOURL 12: Making the Most of     Existing Objects</b>	<b>159</b>
Complex for Loops	106	The Power of Inheritance	159
Testing Your Computer Speed	107	Inheriting Behavior and Attributes	160
		Overriding Methods	161
<b>PART III: Working with Information     in New Ways</b>		Establishing Inheritance	161
		Using this and super in a Subclass	162
<b>HOURL 9: Storing Information with Arrays</b>	<b>111</b>	Working with Existing Objects	163
Creating Arrays	112	Storing Objects of the Same Class in Array Lists	164
Using Arrays	113	Looping Through an Array List	166
Multidimensional Arrays	115	Creating a Subclass	168
Sorting an Array	116		
Counting Characters in Strings	118	<b>PART IV: Moving into Advanced Topics</b>	
		<b>HOURL 13: Storing Objects in     Data Structures</b>	<b>173</b>
<b>HOURL 10: Creating Your First Object</b>	<b>125</b>	Array Lists	174
How Object-Oriented Programming Works	125	Hash Maps	180
Objects in Action	127		
What Objects Are	128		
Understanding Inheritance	129		

<b>HOURL 14: Handling Errors in a Program</b>	<b>187</b>	Buttons	242
Exceptions	188	Labels and Text Fields	245
Catching Exceptions in a try-catch Block	189	Check Boxes	246
Catching Several Different Exceptions	192	Combo Boxes	247
Handling Something After an Exception	194	Text Areas	248
Throwing Exceptions	194	Panels	251
Ignoring Exceptions	196	Creating Your Own Component	251
Exceptions That Don't Need catch	197	<b>HOURL 18: Laying Out a User Interface</b>	<b>259</b>
Throwing and Catching Exceptions	197	Using Layout Managers	259
<b>HOURL 15: Creating a Threaded Program</b>	<b>203</b>	The GridLayout Manager	261
Threads	203	The BorderLayout Manager	262
Slowing Down a Program	204	The BorderLayout Manager	263
Creating a Thread	204	Separating Components with Insets	264
Working with Threads	208	Laying Out an Application	265
The class Declaration	209	<b>HOURL 19: Responding to User Input</b>	<b>273</b>
Setting Up Variables	209	Getting Your Programs to Listen	273
The Constructor	210	Setting Up Components to Be Heard	274
Catching Errors as You Set Up URLs	211	Handling User Events	275
Starting the Thread	211	Check Box and Combo Box Events	276
Running the Thread	212	Keyboard Events	276
Handling Mouse Clicks	213	Enabling and Disabling Components	278
Displaying Revolving Links	213	Completing a Graphical Application	279
Stopping a Thread	216	<b>PART VI: Writing Internet Applications</b>	
<b>HOURL 16: Using Inner Classes and Closures</b>	<b>221</b>	<b>HOURL 20: Reading and Writing Files</b>	<b>291</b>
Inner Classes	222	Streams	291
Anonymous Inner Classes	225	Files	292
Closures	229	Reading Data from a Stream	293
<b>PART V: Programming a Graphical User Interface</b>		Buffered Input Streams	297
<b>HOURL 17: Building a Simple User Interface</b>	<b>237</b>	Writing Data to a Stream	299
Swing and the Abstract Windowing Toolkit	237	Reading and Writing Configuration Properties	302
Using Components	238	<b>HOURL 21: Using Java 9's New HTTP Client</b>	<b>309</b>
Windows and Frames	238	Java Modules	309
		Making an HTTP Request	310
		Saving a File from the Web	314
		Posting Data on the Web	317

<b>HOURL 22: Creating Java2D Graphics</b>	<b>325</b>	Creating a New Project	400
Using the Font Class	325	Creating a New Java Class	402
Using the Color Class	326	Running the Application	404
Creating Custom Colors	327	Fixing Errors	404
Drawing Lines and Shapes	328		
Drawing Lines	328	<b>APPENDIX B: Where to Go from Here:</b>	
Drawing Rectangles	329	<b>Java Resources</b>	<b>407</b>
Drawing Ellipses and Circles	330	Other Books to Consider	407
Drawing Arcs	330	Oracle's Official Java Site	408
Baking a Pie Graph	331	Java Class Documentation	408
		Other Java Websites	408
<b>HOURL 23: Creating Minecraft Mods</b>		This Book's Official Site	408
<b>with Java</b>	<b>341</b>	Workbench	409
Setting Up a Minecraft Server	342	Slashdot	409
Fixing Problems Running the Server	344	Other Java Weblogs	409
Connecting to the Server	346	InformIT	409
Fixing a Server Connection Problem	347	Stack Overflow	409
Creating Your First Mod	348	<i>JavaWorld</i> Magazine	409
Teaching Zombies to Ride Horses	356	Developer.com's Java Directory	410
Finding All Mobs (and Killing Them)	362	Java Meetings	410
Writing a Mod that Can Build Things	366	Job Opportunities	410
		<b>APPENDIX C: This Book's Website</b>	<b>411</b>
<b>HOURL 24: Writing Android Apps</b>	<b>375</b>	<b>APPENDIX D: Fixing a Problem with the</b>	
Introduction to Android	375	<b>Android Studio Emulator</b>	<b>413</b>
Creating an Android App	377	Problems Running an App	413
Exploring a New Android Project	378	Install HAXM in Android Studio	414
Creating an App	380	Install HAXM on Your Computer	415
Setting Up an Android Emulator	382	Checking BIOS Settings	417
Running the App	383	<b>APPENDIX E: Fixing Package Not</b>	
Designing a Real App	385	<b>Visible Errors in NetBeans</b>	<b>419</b>
Organizing Resources	386	Adding Module Info	419
Configuring the App's Manifest File	387		
Designing a User Interface	388	<b>Index</b>	<b>421</b>
Writing Java Code	390		
<b>APPENDICES</b>			
<b>APPENDIX A: Using the NetBeans</b>			
<b>Integrated Development Environment</b>	<b>399</b>		
Installing NetBeans	399		



# Accessing the Free Web Edition

Your purchase of this book in any format, print or electronic, includes access to the corresponding Web Edition, which provides several special features to help you learn:

- ▶ The complete text of the book online
- ▶ Updates and corrections as they become available

The Web Edition can be viewed on all types of computers and mobile devices with any modern web browser that supports HTML5.

To get access to the Web Edition of *Sams Teach Yourself Java in 24 Hours, Eighth Edition*, all you need to do is register this book:

- 1.** Go to [www.informit.com/register](http://www.informit.com/register)
- 2.** Sign in or create a new account
- 3.** Enter ISBN: 9780672337949
- 4.** Answer the questions as proof of purchase

The Web Edition will appear under the Digital Purchases tab on your Account page.

Click the Launch link to access the product.

# 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 *Soap Opera Digest* magazine.

Because of my research, I've learned that if you have picked up this book and turned to this 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, server programming, and Android 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 companies around the world.

This book is aimed at non-programmers, new programmers who think they hate this stuff, and experienced programmers who want to get up to speed swiftly with Java. It uses Java 9, the latest and greatest 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, connect to web services, run on an Android phone or tablet, and more.

This language turns up in some amazing places. One of them is Minecraft, the gaming phenomenon written entirely in Java. (In this book you learn how to create Java programs that run in that game alongside creepers and zombie pigmen!)

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 Internet computing, graphical user interface design, app creation, 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.

---

# HOUR 1

## Becoming a Programmer

---

### This Hour's To-Do List:

- ▶ Find out the reasons to learn Java.
- ▶ Discover how programs work.
- ▶ Select a Java development tool.
- ▶ Get ready to write your first program.

You've probably heard that computer programming is insanely difficult. It requires a degree in computer science, thousands of dollars in computer hardware and software, a keen analytical mind, the patience of Job, and a strong liking for caffeinated drinks.

Aside from the part about caffeine, you heard wrong. Programming is easier than you might think, despite what programmers have been telling people for years to make it easier for us to land high-paying jobs.

This is a great time to learn programming. Countless programming tools are being made available as free downloads on the Web, and thousands of programmers distribute their work as open source so other people can examine how the software was written, fix errors, and contribute improvements. In a recovering economy, many companies are hiring programmers.

It's a great time to learn Java, because the language is everywhere. Billions of mobile devices use Android, an operating system whose apps are all written in Java. If you have an Android phone, you've been enjoying the work of Java programmers every time you look up a movie, rock out on streaming radio, or sling an antagonistic avian at a poorly built fortress of swine.

This book aims to teach Java programming to three kinds of people:

1. Nervous novices who never tried to program before
2. Bitter beginners who tried programming but hated it like Lord Voldemort hates orphaned British schoolchildren
3. Impatient intellectuals who know another programming language and want to get up to speed quickly on Java

To achieve this goal, this book uses the English language as much as possible instead of technical jargon or obscure acronyms. All new programming terms are thoroughly explained as they are introduced.

If I've succeeded, you will finish this book with enough programming skills to be a danger to yourself and others. You'll be able to write programs, plunge into programming classes and books with more confidence, and learn new languages more easily. (Programming languages, to be clear. This book won't help you master Spanish, Esperanto, or Klingon.)

You also will have skills with Java, the most widely used programming language on the planet.

The first hour of this book provides an introduction to programming and guidance on setting up your computer so you can use it to write and run Java programs.

## Choosing a Language

If you're comfortable enough with a computer to prepare a nice-looking résumé, balance a checkbook, or share your vacation photos on Instagram, you can create computer software.

The key to learning how to program is to start with the right language. The programming language you choose often depends on the tasks you want to accomplish. Each language has strengths and weaknesses. Back in my day, young whippersnappers, people learned to program with the BASIC language because it was created with beginners in mind.

### NOTE

---

The BASIC language was invented to be easy for students to learn (the B in BASIC stands for Beginner's). The downside to using some form of BASIC is that it's easy to fall into sloppy programming habits with the language.

---

The most popular language that employs BASIC today is Visual Basic, a programming language from Microsoft that has moved far beyond its roots. VB, as it also is called, is designed for creating programs to run on computers and mobile devices that use the Windows operating system. Another popular language is PHP, a scripting language for creating websites. Other widely used languages you may have heard about include C++, Ruby, Javascript, and Python.

Each of these languages has its adherents, but the most widely taught in computer science classes at the high school and collegiate level is Java.

The Java programming language, which is offered by Oracle, is more difficult to learn than some other languages such as VB and PHP, but it's a great starting place for several reasons. One advantage of learning Java is that you can use it across a variety of operating systems and computing environments. Java programs can be desktop software, web applications, web servers,

Android apps, and more, running on Windows, Mac, Linux, and other operating systems. This versatility is referenced by the ambitious early Java slogan “Write once, run anywhere.”

## NOTE

---

Early Java programmers had a less flattering slogan: “Write once, debug everywhere.” The language has come a long way, baby, since the first version was released in 1996.

---

Another important advantage is that Java requires a highly organized approach for getting programs to work. You must be particular about how you write programs and how they store and alter data.

When you start writing Java programs, you might not see the language’s persnickety behavior as an advantage. You could tire of writing a program and having several errors to fix before the program even can be run. The benefit of this extra effort is that the software you create is more reliable, useful, and error-free.

In the coming hours, you learn all of Java’s rules and the pitfalls to avoid.

Java was invented by the Canadian computer scientist James Gosling as a better way to create computer programs. While working at Sun Microsystems in 1991, Gosling was unhappy with the way the C++ programming language was performing on a project, so he created a new language that did the job better. It’s a matter of contentious debate whether Java is superior to other programming languages, of course, but the success of the language demonstrates the strength of his initial design. Fifteen billion devices across the world are running Java, a number so amazing I’m going to repeat it. Fifteen billion! More than 1,000 books have been published about the language since its introduction. (This is my twentieth.)

Regardless of whether Java is the best language, it definitely is a great language to learn. You get your first chance to try out Java during Hour 2, “Writing Your First Program.”

Learning one programming language makes it much easier to learn subsequent languages. Many are similar to each other, so you aren’t starting from scratch when you plunge into a new one. For instance, many C++ and Smalltalk programmers find it fairly easy to learn Java because Java borrows ideas from those earlier languages. Similarly, C# adopts many ideas from Java, so it’s easier to pick up for Java programmers.

## NOTE

---

C++ is mentioned several times this hour, so you might be tripping over the term, wondering what it means—and how it’s pronounced. C++ is pronounced “C-Plus-Plus,” and it’s a programming language developed by Danish computer scientist Bjarne Stroustrup at Bell Laboratories. C++ is an enhancement of the C programming language, hence the Plus-Plus part of the name. Why not just call it C+? The Plus-Plus part is a computer programming joke you’ll understand later in this book.

---

## Telling the Computer What to Do

A computer program, also called software, is a way to tell a computer to perform a task. Everything that the computer does, from booting up to shutting down, is done by a program. Mac OS X is a program; Minecraft is a program; the driver software that controls your printer is a program; even the dreaded blue screen of death on a crashed Windows PC is a program.

Computer programs are made up of a list of commands the computer handles in a specific order when the program is run. Each command is called a statement.

If your house had its own butler and you were a control freak with a Type-A personality, you could give your servant a detailed set of instructions to follow every day, like this:

Dear Mr. Jeeves,

Please take care of these errands for me while I'm out asking Congress for a bailout:

Item 1: Vacuum the living room.

Item 2: Go to the store.

Item 3: Pick up soy sauce, wasabi, and as many California sushi rolls as you can carry.

Item 4: Return home.

Sincerely, your lord and master,

Bertie Wooster

If you tell a human butler what to do, there's a certain amount of leeway in how your requests are fulfilled. If California rolls aren't available, Jeeves could bring Boston rolls home instead.

Computers don't do leeway. They follow instructions literally. The programs that you write are followed precisely, one instruction at a time.

The following example is a three-line computer program, written in BASIC. Take a look at it, but don't worry too much about what each line is supposed to mean.

```
1 PRINT "Hey Tom, it's Bob from the office down the hall."  
2 PRINT "It's good to see you buddy, how've you been?"  
3 INPUT A$
```

Translated into English, this program is equivalent to giving a computer the following to-do list:

Dear personal computer,

Item 1: Display the message, "Hey Tom, it's Bob from the office down the hall."

Item 2: Ask the question, "It's good to see you buddy, how've you been?"

Item 3: Give the user a chance to answer the question.

Sincerely, your lord and master,

Ima Coder

Each line in a computer program is called a *statement*. A computer handles each statement in a program in a specific order, in the same way that a cook follows a recipe or Mr. Jeeves the butler follows the orders of Bertie Wooster. In BASIC, the line numbers are used to put the statements in the correct order. Other languages such as Java do not use line numbers, favoring different ways to tell the computer how to run a program.

Because of the way programs function, you can't blame the computer when something goes wrong as your program runs. The computer is doing exactly what you told it to do, so the blame for any errors usually lies with the programmer.

That's the bad news. The good news is you can't do any permanent harm. No computers will be injured as you learn to program in Java.

## How Programs Work

The collection of statements that make up a computer program is called its *source code*.

Most computer programs are written in the same way that you write an email—by typing each statement into a text window. Some programming tools come with their own source code editor and others can be used with any text-editing software.

When you have finished writing a computer program, you save the file to disk. Computer programs often have their own filename extension to indicate what type of file they are. Java programs must have the extension `.java`, as in `Calculator.java`.

### NOTE

---

Computer programs should be prepared as text files with no special formatting. Notepad, a text editor that comes with Windows, saves all files as unformatted text. You also can use TextEdit on Macs or the vi editor or emacs on Linux systems to create text files without formatting. An easier solution is coming up later this hour.

---

To run a program you have saved as a file, you need some help. The kind of help required depends on the programming language you're using. Some languages require an interpreter to run their programs. The interpreter examines each line of a computer program and executes that line, then proceeds to the next line. Many versions of BASIC are interpreted languages.



The biggest advantage of interpreted languages is that they are faster to test. When you are writing a BASIC program, you can try it out immediately, fix errors, and try again. The primary disadvantage is that interpreted languages run slower than other programs. Each line has to be translated into instructions the computer can run, one line at a time.

Other programming languages require a compiler. The compiler takes a program and translates it into a form that the computer can understand. It also makes the program run as efficiently as possible. The compiled program can be run directly without the need for an interpreter.

Compiled programs run more quickly than interpreted programs but take more time to test. You have to write your program and compile the whole thing before trying it out. If you find an error and fix it, you must compile the program again.

Java is unusual because it requires both a compiler and an interpreter. The compiler converts the statements that make up the program into bytecode. Once this bytecode has been created successfully, it can be run by an interpreter called the Java Virtual Machine.

The Java Virtual Machine, also called a JVM, is the thing that makes it possible for the same Java program to run without modification on different operating systems and different kinds of computing devices. The virtual machine turns bytecode into instructions that a particular device's operating system can execute.

## NOTE

---

Java 9 introduces a new tool called JShell that acts like an interpreter, running a Java statement right when it is typed in. JShell works by putting the statement into a Java program, compiling that program into bytecode, and running it. This is a useful tool for learning and testing.

---

## When Programs Don't Work

Many new programmers become discouraged when they start to test their programs. Errors appear everywhere. Some of these are syntax errors, which are identified by the computer as it looks at the program and becomes confused by the way a statement has been written. Other errors are logic errors, which only are noticed by the programmer as the program is being tested (or might be overlooked entirely). Logic errors often cause it to do something unintended.

As you begin writing your own programs, you become well acquainted with errors. They're a natural part of the process. Programming errors are called *bugs*, a term that dates back a century or more to describe errors in technical devices.

The process of fixing errors also has its own term: *debugging*.

It's no coincidence that there are so many ways to describe errors. You get a lot of debugging experience as you learn programming—whether you want it or not.

## NOTE

---

One of the first computer bugs was discovered in 1947 by a team that included the American computer scientist Grace Hopper. Hopper was testing a computer at Harvard when a relay malfunctioned. The cause wasn't a software problem—it was an actual bug! A team member debugged the computer by removing a dead moth and taped it into a logbook with the note, "First actual case of bug being found." The bug and logbook page can be viewed at [www.doncio.navy.mil/CHIPS/ArticleDetails.aspx?id=3489](http://www.doncio.navy.mil/CHIPS/ArticleDetails.aspx?id=3489).

---

## Choosing a Java Programming Tool

To start writing Java programs, you must have a Java programming tool. Several such programs are available for Java, including the simple Java Development Kit and the more sophisticated Eclipse, IntelliJ IDEA, and NetBeans. The latter three tools are each an integrated development environment (IDE), a powerful tool used by professional programmers to get work done.

Whenever Oracle releases a new version of Java, the first tool that supports it is the Java Development Kit (JDK).

To create the programs in this book, you must use JDK version 9 or a programming tool that works on top of it. The JDK is a set of free command-line tools for creating Java software. It lacks a graphical user interface, so if you have never worked in a non-graphical environment such as the Windows command prompt or Linux command-line interface, you will find it challenging to use the JDK.

The NetBeans IDE, also offered for free by Oracle, is a much easier way to write and test Java code than the JDK. NetBeans includes a graphical user interface, source code editor, user interface designer, and project manager. It works in complement to the JDK, running it behind the scenes, so you must have both tools on your system when you begin developing Java programs.

Most of the programs in this book were created with NetBeans, which you can download and install separately from the JDK. You can use other Java tools as long as they support JDK 9.

## NOTE

---

You don't have to use NetBeans in this book. If you can use the JDK or another tool to create, compile, and run a program, those tasks are all that most projects require. NetBeans is covered because for readers of past editions it has proven easier than the JDK. I use NetBeans for most of my Java programming.

---