IN THIS CHAPTER

- What Java is
- Where Java came from
- Why Java is so cool
- Three ways to write computer code

# Chapter 1
# All about Java



Say what you want about computers. As far as I'm concerned, computers are good for just two simple reasons:

- **When computers do work, they feel no resistance, no stress, no boredom, and no fatigue.** Your computer can work 24/7 making calculations for `www.climateprediction.net` — a distributed computing project to model the world's climate change. Or, have your computer crunch numbers for `Rosetta@home` — a site that models proteins to help cure major illnesses. Will you feel sorry for my computer because it's working so hard? Will the computer complain? No.

  You can make demands, give the computer its orders, and crack the whip. Will you (or should you) feel the least bit guilty? Not at all.

- **Computers move ideas, not paper.** Not long ago, whenever you wanted to send a message to someone, you hired a messenger. The messenger mounted a horse and delivered your message personally. The message was recorded on paper or parchment or a clay tablet or whatever other physical medium was available at the time.

  This whole process seems wasteful now, but that's only because you and I are sitting comfortably in the electronic age. Messages are ideas, and physical objects like ink, paper, and horses have little or nothing to do with real ideas; they're just temporary carriers of ideas (even though people used them for

several centuries to carry ideas). Nevertheless, the ideas themselves are paperless, horseless, and messengerless.

The neat thing about computers is that they carry ideas efficiently. They carry nothing but the ideas, a couple of photons, and some electrical power. They do this with no muss, no fuss, and no extra physical baggage.

When you start dealing efficiently with ideas, something very nice happens: Suddenly, all overhead is gone. Instead of pushing paper and trees, you're pushing numbers and concepts. Without the overhead, you can do things much faster and do things that are far more complex than ever.

# What You Can Do with Java

It would be nice if all this complexity were free, but, unfortunately, it isn't. Someone has to think hard and decide exactly what to ask the computer to do. After that thinking takes place, someone has to write a set of instructions for the computer to follow.

Given the current state of affairs, you can't write these instructions in English or any other language that people speak. Science fiction is filled with stories about people who make simple requests of robots and get back disastrous, unexpected results. English and other such languages are unsuitable for communication with computers, for several reasons:

- **An English sentence can be misinterpreted.** "No parking. Violators will be towed away at the owner's expense." So, the parking space's owner will pay to tow my car? That's not so bad!
- **It's difficult to weave a complicated command in English.** "Join flange A to protuberance B, making sure to connect only the outermost lip of flange A to the larger end of the protuberance B while joining the middle and inner lips of flange A to grommet C."
- **English sentences have lots of extra baggage.** "Sentences have unneeded words."
- **English can be difficult to interpret.** "John Wiley & Sons, Inc. shall pay net earnings to the Author ('Barry Burd') or the Author's assigns upon submittal of *Java For Dummies,* 9th Edition ('the Work') minus prepaid and accrued expenses and deferred charges, either directly or indirectly, within the meaning of section 4942(j)(3) or 4965(k)(5) for calendar year 2024."

To tell a computer what to do, you have to use a special language to write terse, unambiguous instructions. A special language of this kind is called a *computer programming language.* A set of instructions written in such a language is called a *program.* When looked at as a big blob, these instructions are called *software* or *code.* Here's what code looks like when it's written in Java:

```
void main() {
    var checkAmount = 1257.63;
    println("Pay to the order of Dr. Barry Burd $" + checkAmount);
}
```

You may argue that ChatGPT and other generative AI tools narrow the gap between informal English and carefully written code. That's true to some extent. But computer code isn't going away anytime soon. Generative AI may hallucinate. And, in some critical applications, hallucinations are unacceptable. Besides, GPT tools don't create themselves. At this very moment (whenever you're reading my book, that is), thousands of people around the world are writing code to make GPT tools faster and more versatile. Programming isn't a dying art. It's a growing endeavor.

# Why You Should Use Java

It's time to celebrate! You've just picked up a copy of *Java For Dummies,* 9th Edition, and you're reading Chapter 1. At this rate, you'll be an expert Java programmer* in no time at all, so rejoice in your eventual success by throwing a big party.

To prepare for the party, I'll bake a cake. I'm lazy, so I'll use a ready-to-bake cake mix. Let me see: Add water to the mix and then add butter and eggs — hey, wait! I just looked at the list of ingredients. What's MSG? And what about propylene glycol? That's used in antifreeze, isn't it?

I'll change plans and make the cake from scratch. Sure, it's a little harder, but that way, I get exactly what I want.

Computer programs work the same way: You can use somebody else's program or write your own. If you use somebody else's program, you use whatever you get. When you write your own program, you can tailor the program especially for your needs.

* In professional circles, a developer's responsibilities are usually broader than those of a programmer. But, in this book, I use the terms *programmer* and *developer* almost interchangeably.

Writing computer code is a big, worldwide industry. Companies do it, freelance professionals do it, hobbyists do it — all kinds of people do it. A typical big company has teams, departments, and divisions that write programs for the company. But you can write programs for yourself or for someone else, for a living or for fun. In a recent estimate, the number of lines of code written each day by programmers in the world exceeds the number of methane molecules on the planet Jupiter.** Take almost anything that can be done with a computer — with the right amount of time, you can write your own program to do it. (Of course, the "right amount of time" may be quite long, but that's not the point. Many interesting and useful programs can be written in hours or even minutes.)

# Gaining Perspective: Where Java Fits In

Here's a brief history:

- **1954–1957: FORTRAN is developed.**

  FORTRAN was the first modern computer programming language. For scientific programming, FORTRAN is a real racehorse. Year after year, FORTRAN is a leading language among computer programmers throughout the world.

- **1959: Grace Hopper at Remington Rand develops the COBOL programming language.**

  The letter *B* in COBOL stands for *Business,* and business is just what COBOL is all about. The language's primary feature is the processing of one record after another, one customer after another, or one employee after another.

  Within a few years after its initial development, COBOL became the most widely used language for business data processing.

  (Fun fact: I once foolishly tried to write a COBOL program to create musical compositions. Talk about picking the wrong tool for the job! It was like brushing my teeth with a nail file.)

- **1960: John McCarthy writes a paper on his new LISP programming language.**

  LISP sets the stage for a style of programming called *functional programming*. For a few decades, functional programming waits in the background while other programming techniques take center stage.

** I made up this fact all by myself.

- **1967: Scientists at the Norwegian Computing Center develop the Simula 67 language.**

  With Simula 67, object-oriented programming is born.

- **1972: Dennis Ritchie at AT&T Bell Labs develops the C programming language.**

  The "look and feel" that you see in this book's examples comes from the C programming language. Code written in C uses curly braces, `if` statements, `for` statements, and other elements.

  In terms of power, you can use C to solve the same problems that you can solve by using FORTRAN or Java or any other modern programming language. The difference between one programming language and another isn't power — the difference is ease and appropriateness of use. That's where the Java language excels.

- **1986: Bjarne Stroustrup (also at AT&T Bell Labs) develops C++.**

  Unlike its C language ancestor, the C++ language supports object-oriented programming. This support represents a huge step forward. (See the next section in this chapter.)

- **May 23, 1995: Sun Microsystems releases its first official version of the Java programming language.**

  Java improves upon the concepts in C++. Java not only supports object-oriented programming but also *enforces the use of* object-oriented programming.

  Additionally, Java is a great general-purpose programming language. A program written in Java runs seamlessly on all major platforms, including Windows, Macintosh, and Linux. With Java, you can write windowed applications, build and explore databases, control handheld devices, and more. Within five short years, the Java programming language has 2.5 million developers worldwide. (I know — I have a commemorative T-shirt to prove it.)

  By 2001, Java has become the number-one language on the world-famous TIOBE Programming Community Index.

- **2007-2017: Java grows up.**

  For a while, updates to Java come very slowly. With disagreements over the implementation of new features, the gap between Java 6 and Java 7 releases is a whopping five years.

  In the meantime, Oracle Corporation purchases Sun Microsystems — a terrific bargain for only $7.4 billion! Then, in 2014, Oracle releases Java 8 — the first Java version with functional programming capabilities.

  For most of the months between 2001 and 2017, Java has maintained its top position on the TIOBE Index.

» **August 2017: Oracle announces its plan to release new versions of Java every six months.**

The release of Java 9 in September 2017 is followed by the rollout of Java 10 in March 2018. Up next is Java 11 in September 2018.

In September 2023, Java 21 is a *l*ong-*t*erm *s*upport (LTS) release. This means that Oracle promises to keep Java 21 running smoothly at least until September 2028. These LTS releases come every two years, so the next rock-solid, take-no-prisoners version of Java is Java 25 in September 2025.

The new release cycle has injected energy into the evolution of the Java programming language.

Java technology powers applications from companies like Microsoft, Uber, LinkedIn, PayPal, Netflix, Airbnb, Google, eBay, Spotify, TripAdvisor, Intel, Pinterest, and Groupon.* The job search site `Monster.com` says:

> "Java is one of the most popular programming languages in use, so it's no surprise it came in as the No. 1 skill tech companies were looking for. According to Oracle, 3 billion mobile phones run Java, along with 125 million TV devices and 89% of desktop computers in the U.S. Java is everywhere and the demand for strong developers is high."**

Well, I'm impressed.

# Three Ways to Write Computer Programs

It's 4 in the morning. I'm dreaming about the history course I failed in high school. The teacher is yelling at me: "You have two days to study for the final exam, but you won't remember to study. You'll forget and feel guilty, guilty, guilty."

Suddenly, the phone rings. I'm awakened abruptly from my deep sleep. (Sure, I disliked dreaming about the history course, but I like being awakened even less.) At first, I drop the telephone on the floor. After fumbling to pick it up, I issue a grumpy, "Hello. Who's this?" A voice answers, "I'm a reporter from the Reuters

---

Sources:

*`https://codegym.cc/groups/posts/771-is-java-still-relevant-what-big-companies-use-it`

**`www.monster.com/career-advice/article/programming-languages-you-should-know`

news agency. I'm writing an article about Java, and I need to know all about the programming language in seven words or less. Can you explain it?"

My mind is too hazy. I can't think. So I say the first thing that comes to my mind and then go back to sleep.

Come morning, I hardly remember the conversation with the reporter. In fact, I don't remember how I answered the question. Did I utter a few obscenities and then go back to sleep?

I put on my robe and rush to my computer. When I visit the Reuters website, I see this enormous headline:

> **Burd Calls Java "A Great Language in Many Different Ways"**

## A word you can use to impress people

Today's vocabulary word is the word *paradigm*.

First things first: How do you pronounce the word *paradigm*? The *g* is silent. You say "PAA-ra-dime."

Next, what in the world does *paradigm* mean? I looked up *paradigm* in a few dictionaries, and none of the definitions impressed me much. To my mind, a *paradigm* is a way of thinking about something. For example, people used to think about the sun orbiting the Earth. But, in 1543, Copernicus proposed that the Earth orbits the sun. The Copernican Revolution was a paradigm shift.

Over the years, several computer programming paradigms have emerged. The following sections describe three of them: imperative programming, object-oriented programming, and functional programming.

## Imperative programming

Once upon a time, almost every programming language was imperative in nature. With an imperative language, your programs consist mostly of "Do this, then do that" commands. Suppose that you run a small business and you have the three employees listed in Table 1-1.

TABLE 1-1 A Table of Employees

| Name | Hours Per Week | Pay |
|---|---|---|
| Alice | 40 | 450.00 |
| Bob | 20 | 200.00 |
| Carol | 35 | 300.00 |

You want each of your employees to work an additional hour every week. In exchange for this, you'll pay each employee $50 more. Here's how you might do it with an imperative style:

```
For each of my employees:
    Get the employee's hours per week,
    add 1 to that number,
    make that the new value of the employee's hours per week.
    Get the employee's current pay,
    add $50 to that amount,
    make that amount the new value of the employee's pay.
```

REMEMBER

What you see here isn't a real computer program. It's an informal, English-language description of the way a real program might work. Since it's not real code, we call it *pseudocode*.

In an *imperative program*, your code is simply a long list of commands. You may set aside some commands to be run separately and give that block of commands a name. But, one way or another, the code's fundamental building blocks are individual commands.

A typical command either gets a stored value or sets a stored value. For example, one command may get an employee's current pay while another command sets the employee's new pay. With imperative programming, each value is loosely connected to all the other values. For example, when your code changes an employee's hours per week, the code happens to change the employee's pay. Other than that, there may be no relationship between hours per week and pay. Each value is an entity on its own.

# Object-oriented programming

I often say that imperative programming is "verbs first" and that object-oriented programming (OOP) is "nouns first." With imperative programming, you begin with verbs like *get* and *add.* With object-oriented programming, you begin by defining the kinds of things that your code deals with.

Here's pseudocode to describe the way an object-oriented program defines an employee in your company:

```
An Employee has:
    a name,
    a number of hours per week,
    a pay amount,
    the following code, called add1Hour:
        Add 1 to my hours per week;
    the following code, called add50Dollars:
        Add $50 to my pay.
```

When you write an object-oriented program, you translate this pseudocode into formal Java code and call that code the *Employee class*. Your program may include code for an `Employee` class, a `Customer` class, a `Supplier` class, and many other classes.

Here's something interesting: An `Employee` has data (the employee's name, hours per week, and pay), but an `Employee` also has code for working with the data (`add1Hour` and `add50Dollars`). In object-oriented programming, each class owns code for working with that class. It's as if you'd raise Alice's salary by saying "Alice, please raise your salary." As strange as that may sound, it's a pivotal feature of OOP. The underlying idea is to keep everything you want to know about an `Employee` (data and code) together in one place.

Notice that the `Employee` class code says nothing about any of the individual employees in Table 1-1. The `Employee` class is only a description of the kinds of information that you'll eventually store for each employee. When you code the `Employee` class, you're not creating any actual employees. You can think of the `Employee` class as a table with nothing but a header row. (See Table 1-2.)

**TABLE 1-2** **A Tabular Representation of the Employee Class**

| Name | Hours per week | Pay | Add one hour | Add $50 pay |
|---|---|---|---|---|
| | | | | |

**REMEMBER**

In object-oriented programming, a class is like a table's header row. A class is the idea behind a certain kind of thing. When I talk about the class of employees, I'm talking about the fact that each employee has a name, a weekly pay amount, a way of adding to the weekly pay, and so on. The pay amounts may be different for different employees, but that doesn't matter. When I talk about a class of things, I'm focusing on the properties that each of the things possesses.

Let's pause for a minute. How do you feel about Table 1-2? Do you like Table 1-2?

Ugh! I don't like Table 1-2. An `Employee` table with no actual employees isn't complete. It's like a day without chocolate. Classes alone don't give object-oriented programming its power.

To complete the picture, assume that you already have Java code that describes the `Employee` class. Now it's time to make use of that code. So you write Java code whose translation into informal English looks something like this:

```
Create an object named Alice from the Employee class. Alice works 40 hours per
   week with pay $450.
Create an object named Bob from the Employee class. Bob works 20 hours per week
   with pay $200.
Create an object named Carol from the Employee class. Carol works 35 hours per
   week with pay $300.
```

You may be uncomfortable calling Alice, Bob, and Carol "objects", but remember that the people themselves aren't objects. It's the information about these people that you call "objects." Since you fashioned each object from the `Employee` class, you call each of these objects an *instance* of the `Employee` class.


REMEMBER

Because Java is an object-oriented programming language, your primary goal is to describe classes and objects. A class is the idea behind a certain kind of thing. An object is a concrete instance of a class. First, you define a class. Then, from the class definition, you construct individual objects.

Having written the code to create three objects, you now have a more complete table. (See Table 1-3.)

**TABLE 1-3** The `Employee` Class with Three Objects

| Name | Hours per week | Pay | Add one hour | Add $50 pay |
|---|---|---|---|---|
| Alice | 40 | 450.00 | 40 + 1 is 41 | 450 + 50 is 500 |
| Bob | 20 | 200.00 | 20 + 1 is 21 | 200 + 50 is 250 |
| Carol | 35 | 300.00 | 35 + 1 is 36 | 300 + 50 is 350 |

Think of the header row in Table 1-3 as the `Employee` class. Then each of the other rows is an object – an instance of the `Employee` class.

Now that you've created three `Employee` instances, you can write code to increase each employee's hour and pay. Here's the pseudocode version:

```
For each employee,
    run that employee's add1Hour code,
    run that employee's add50Dollars code.
```

Once you've created a class and its objects, the actual work of changing your employees' hours and pay is straightforward and concise. And that, my friends, is what object-oriented programming is all about.

Wait! What do I hear you saying to yourself? You say this description of object-oriented programming is vague. You don't really understand it yet. Okay. I admit it. I can't teach you all about OOP in Chapter 1. There's a lot more detail in the chapters to come. Just remember that Java is, first and foremost, an object-oriented programming language. In Java, everything is part of the object-oriented framework. To my mind, Java's implementation of the object-oriented paradigm is the best there is.

TIP

With a name like "object-oriented programming," you'd expect the word "object" to appear several times on each page of this book. But that's not the case. Each object that you create with the `Employee` class is an "*instance* of the `Employee` class," also known as an "`Employee` class *instance*" or just "`Employee` *instance*." If you use the term "`Employee` object" instead of "`Employee` instance," people will know what you mean. And sometimes, you can say "an `Employee`" when you really mean "an `Employee` instance." But your best bet is to remember that each object you construct using the `Employee` class is called an `Employee` instance.

# Functional programming

*Functional programming* has several moving parts. But, in this chapter, you can think of functional programming as an assembly line where data travels from one station to another. Each station makes a new bundle of data from the old bundle and sends the new bundle on its way. One station, usually called `forEach` or `map`, takes a bunch of things along with a piece of code and applies the code to each of the things.

You can use `forEach` to increase the hours and pay for all your employees. Here's some pseudocode to show you how it works:

```
Start with the list of employees,
then apply forEach with the code add1Hour to the list,
then apply forEach with the code add50Dollars to the list.
```

By the way, as this chapter has progressed, I've been hearing from Alice. Table 1-3 says she'll have to work more than 40 hours a week, and Alice is upset about that. Why don't you change your plans? Consider adding an hour to each employee's work week but do this only for employees who aren't already working 40 hours.

Functional programming has something called `filter`, which accepts only items that satisfy certain criteria. Here's how `filter` can keep Alice from working overtime:

```
Start with the list of employees,
then filter for employees who currently work less than 40 hours,
then apply forEach with the code add1Hour to the filtered list,
then apply forEach with the code add50Dollars to the filtered list.
```

But wait! When you keep Alice's hours at 40, will your employees be working enough hours to keep your business afloat? To find out, you pull another functional programming trick out of your hat. This trick is called `reduce`. It takes a piece of code and combines a list's values using that code. Here's what you get:

```
Start with the list of employees,
then filter for employees who currently work less than 40 hours,
then apply forEach with the code add1Hour to the filtered list,
then apply forEach with the code add50Dollars to the filtered list,
then apply forEach with code to retrieve hours per week from the filtered list,
then apply reduce with the code to do addition to the filtered list of
   employees' hours per week.
```

The diagram in Figure 1-1 shows how this functional programming strategy resembles an assembly line.

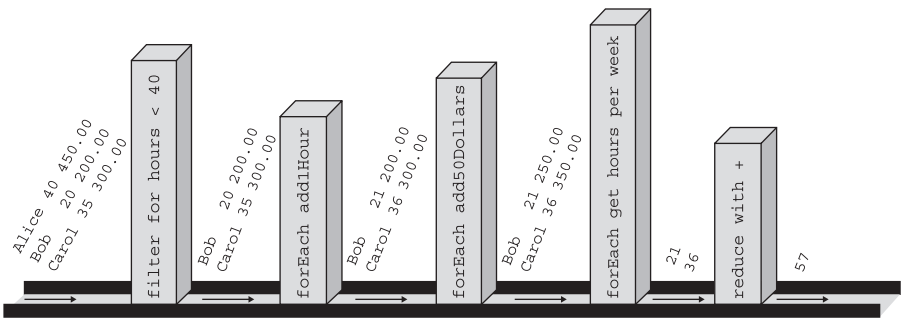


**FIGURE 1-1:** The functional programming assembly line.

Like my rambling about OOP in the previous section, this section's description of functional programming is decidedly vague. But that's okay because Chapter 12 has complete examples to show how functional programming works in Java.

## What's Next?

This chapter is filled with general descriptions of things. A general description is good when you're just getting started, but you don't truly understand things until you get to know some specific info, as laid out in the next several chapters.

So please, turn the page. The next chapter can't wait for you to read it.