Hello and welcome to *Teach Yourself Java in 21 Days*! Starting today and for the next three weeks you'll learn all about the Java language and how to use it to create applets, as well as how to create stand-alone Java applications that you can use for just about anything.

NEW TERM An *applet* is a dynamic and interactive program that can run inside a Web page displayed by a Java-capable browser such as HotJava or Netscape 2.0.

The *HotJava browser* is a World Wide Web browser used to view Web pages, follow links, and submit forms. It can also download and play applets on the reader's system.

That's the overall goal for the next three weeks. Today, the goals are somewhat more modest, and you'll learn about the following:

- ☐ What exactly Java and HotJava are, and their current status
- ☐ Why you should learn Java—its various features and advantages over other programming languages
- ☐ Getting started programming in Java—what you'll need in terms of software and background, as well as some basic terminology
- ☐ How to create your first Java programs—to close this day, you'll create both a simple Java application and a simple Java applet!

# What Is Java?

Java is an object-oriented programming language developed by Sun Microsystems, a company best known for its high-end Unix workstations. Modeled after C++, the Java language was designed to be small, simple, and portable across platforms and operating systems, both at the source and at the binary level (more about this later).

Java is often mentioned in the same breath as HotJava, a World Wide Web browser from Sun like Netscape or Mosaic (see Figure 1.1). What makes HotJava different from most other browsers is that, in addition to all its basic Web features, it can also download and play applets on the reader's system. Applets appear in a Web page much in the same way as images do, but unlike images, applets are dynamic and interactive. Applets can be used to create animations, figures, or areas that can respond to input from the reader, games, or other interactive effects on the same Web pages among the text and graphics.

Although HotJava was the first World Wide Web browser to be able to play Java applets, Java support is rapidly becoming available in other browsers. Netscape 2.0 provides support for Java applets, and other browser developers have also announced support for Java in forthcoming products.


**Figure 1.1.**
*The HotJava browser.*

To create an applet, you write it in the Java language, compile it using a Java compiler, and refer to that applet in your HTML Web pages. You put the resulting HTML and Java files on a Web site much in the same way that you make ordinary HTML and image files available. Then, when someone using the HotJava browser (or other Java-aware browser) views your page with the embedded applet, that browser downloads the applet to the local system and executes it, and then the reader can view and interact with your applet in all its glory (readers using other browsers won't see anything). You'll learn more about how applets, browsers, and the World Wide Web work together further on in this book.

The important thing to understand about Java is that you can do so much more with it besides create applets. Java was written as a full-fledged programming language in which you can accomplish the same sorts of tasks and solve the same sorts of problems that you can in other programming languages, such as C or C++. HotJava itself, including all the networking, display, and user interface elements, is written in Java.

# Java's Past, Present, and Future

The Java language was developed at Sun Microsystems in 1991 as part of a research project to develop software for consumer electronics devices—television sets, VCRs, toasters, and the other sorts of machines you can buy at any department store. Java's goals at that time were to be small, fast, efficient, and easily portable to a wide range of hardware devices. It is those same goals that made Java an ideal language for distributing executable programs via the World Wide Web, and also a general-purpose programming language for developing programs that are easily usable and portable across different platforms.

The Java language was used in several projects within Sun, but did not get very much commercial attention until it was paired with HotJava. HotJava was written in 1994 in a matter of months, both as a vehicle for downloading and running applets and also as an example of the sort of complex application that can be written in Java.

At the time this book is being written, Sun has released the beta version of the Java Developer's Kit (JDK), which includes tools for developing Java applets and applications on Sun systems running Solaris 2.3 or higher for Windows NT and for Windows 95. By the time you read this, support for Java development may have appeared on other platforms, either from Sun or from third-party companies.

Note that because the JDK is currently in beta, it is still subject to change between now and when it is officially released. Applets and applications you write using the JDK and using the examples in this book may require some changes to work with future versions of the JDK. However, because the Java language has been around for several years and has been used for several projects, the language itself is quite stable and robust and most likely will not change excessively. Keep this beta status in mind as you read through this book and as you develop your own Java programs.

Support for playing Java programs is a little more confusing at the moment. Sun's HotJava is not currently included with the Beta JDK; the only available version of HotJava is an older alpha version, and, tragically, applets written for the alpha version of Java do not work with the beta JDK, and vice versa. By the time you read this, Sun may have released a newer version of HotJava which will enable you to view applets.

The JDK does include an application called appletviewer that allows you to test your Java applets as you write them. If an applet works in the appletviewer, it should work with any Java-capable browser. You'll learn more about applet viewer later today.

What's in store for the future? In addition to the final Java release from Sun, other companies have announced support for Java in their own World Wide Web browsers. Netscape Communications Corporation has already incorporated Java capabilities into the 2.0 version of their very popular Netscape Navigator Web browser—pages with embedded Java applets can be viewed and played with Netscape. With support for Java available in as popular a browser as Netscape,

tools to help develop Java applications (debuggers, development environments, and so on) most likely will be rapidly available as well.

# Why Learn Java?

At the moment, probably the most compelling reason to learn Java—and probably the reason you bought this book—is that HotJava applets are written in Java. Even if that were not the case, Java as a language has significant advantages over other languages and other programming environments that make it suitable for just about any programming task. This section describes some of those advantages.

## Java Is Platform-Independent

Platform independence is one of the most significant advantages that Java has over other programming languages, particularly for systems that need to work on many different platforms. Java is platform-independent at both the source and the binary level.

NEW TERM *Platform-independence* is a program's capability of moving easily from one computer system to another.

At the source level, Java's primitive data types have consistent sizes across all development platforms. Java's foundation class libraries make it easy to write code that can be moved from platform to platform without the need to rewrite it to work with that platform.

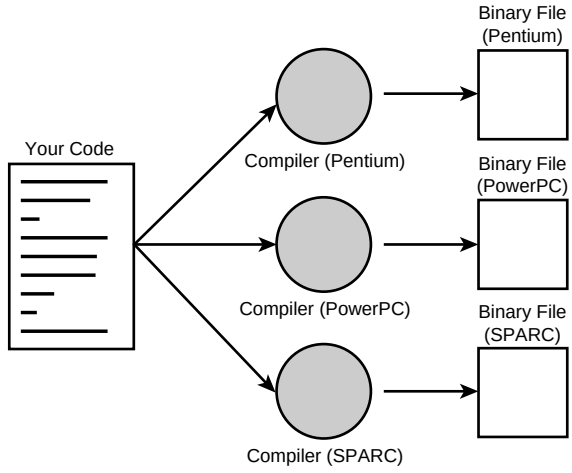
Platform-independence doesn't stop at the source level, however. Java binary files are also platform-independent and can run on multiple problems without the need to recompile the source. How does this work? Java binary files are actually in a form called bytecodes.

NEW TERM *Bytecodes* are a set of instructions that looks a lot like some machine codes, but that is not specific to any one processor.

Normally, when you compile a program written in C or in most other languages, the compiler translates your program into machine codes or processor instructions. Those instructions are specific to the processor your computer is running—so, for example, if you compile your code on a Pentium system, the resulting program will run only on other Pentium systems. If you want to use the same program on another system, you have to go back to your original source, get a compiler for that system, and recompile your code. Figure 1.2 shows the result of this system: multiple executable programs for multiple systems.

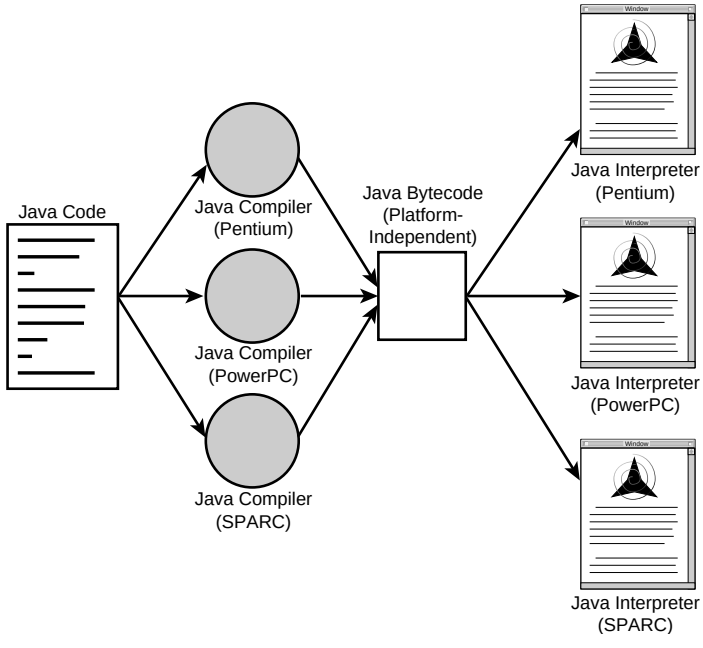
Things are different when you write code in Java. The Java development environment has two parts: a Java compiler and a Java interpreter. The Java compiler takes your Java program and instead of generating machine codes from your source files, it generates bytecodes.

**Figure 1.2.**
*Traditional compiled programs.*




To run a Java program, you run a program called a bytecode interpreter, which in turn executes your Java program (see Figure 1.3). You can either run the interpreter by itself, or—for applets—there is a bytecode interpreter built into HotJava and other Java-capable browsers that runs the applet for you.

**Figure 1.3.**
*Java programs.*



Why go through all the trouble of adding this extra layer of the bytecode interpreter? Having your Java programs in bytecode form means that instead of being specific to any one system, your programs can be run on any platform and any operating or window system as long as the Java interpreter is available. This capability of a single binary file to be executable across platforms is crucial to what enables applets to work, because the World Wide Web itself is also platform-independent. Just as HTML files can be read on any platform, so applets can be executed on any platform that is a Java-capable browser.

The disadvantage of using bytecodes is in execution speed. Because system-specific programs run directly on the hardware for which they are compiled, they run significantly faster than Java bytecodes, which must be processed by the interpreter. For many Java programs, the speed may not be an issue. If you write programs that require more execution speed than the Java interpreter can provide, you have several solutions available to you, including being able to link native code into your Java program or using tools to convert your Java bytecodes into native code. Note that by using any of these solutions, you lose the portability that Java bytecodes provide. You'll learn about each of these mechanisms on Day 20.

# Java Is Object-Oriented

To some, object-oriented programming (OOP) technique is merely a way of organizing programs, and it can be accomplished using any language. Working with a real object-oriented language and programming environment, however, enables you to take full advantage of object-oriented methodology and its capabilities of creating flexible, modular programs and reusing code.

Many of Java's object-oriented concepts are inherited from C++, the language on which it is based, but it borrows many concepts from other object-oriented languages as well. Like most object-oriented programming languages, Java includes a set of class libraries that provide basic data types, system input and output capabilities, and other utility functions. These basic classes are part of the Java development kit, which also has classes to support networking, common Internet protocols, and user interface toolkit functions. Because these class libraries are written in Java, they are portable across platforms as all Java applications are.

You'll learn more about object-oriented programming and Java tomorrow.

# Java Is Easy to Learn

In addition to its portability and object-orientation, one of Java's initial design goals was to be small and simple, and therefore easier to write, easier to compile, easier to debug, and, best of all, easy to learn. Keeping the language small also makes it more robust because there are fewer chances for programmers to make difficult-to-find mistakes. Despite its size and simple design, however, Java still has a great deal of power and flexibility.

Java is modeled after C and C++, and much of the syntax and object-oriented structure is borrowed from the latter. If you are familiar with C++, learning Java will be particularly easy for you, because you have most of the foundation already.

Although Java looks similar to C and C++, most of the more complex parts of those languages have been excluded from Java, making the language simpler without sacrificing much of its power. There are no pointers in Java, nor is there pointer arithmetic. Strings and arrays are real objects in Java. Memory management is automatic. To an experienced programmer, these omissions may be difficult to get used to, but to beginners or programmers who have worked in other languages, they make the Java language far easier to learn.

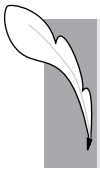
# Getting Started with Programming in Java

Enough background! Let's finish off this day by creating two real Java programs: a stand-alone Java application and an applet that you can view in either in the appletviewer (part of the JDK) or in a Java-capable browser. Although both these programs are extremely simple, they will give you an idea of what a Java program looks like and how to compile and run it.

## Getting the Software

In order to write Java programs, you will, of course, need a Java development environment. At the time this book is being written, Sun's Java Development Kit provides everything you need to start writing Java programs. The JDK is available for Sun SPARC systems running Solaris 2.2 or higher and for Windows NT and Windows 95. You can get the JDK from several places:

- The CD-ROM that came with this book contains the full JDK distribution. See the CD information for installation instructions.
- The JDK can be downloaded from Sun's Java FTP site at `ftp://java.sun.com/pub/` or from a mirror site (`ftp://www.blackdown.org/pub/Java/pub/is one`).



**Note:** The Java Development Kit is currently in beta release. By the time you read this, The JDK may be available for other platforms, or other organizations may be selling Java development tools as well.

Although Netscape and other Java-aware browsers provide an environment for playing Java applets, they do not provide a mechanism for developing Java applications. For that, you need separate tools—merely having a browser is not enough.

# Applets and Applications

Java applications fall into two main groups: applets and applications.

Applets, as you have learned, are Java programs that are downloaded over the World Wide Web and executed by a Web browser on the reader's machine. Applets depend on a Java-capable browser in order to run (although they can also be viewed using a tool called the appletviewer, which you'll learn about later today).

Java applications are more general programs written in the Java language. Java applications don't require a browser to run, and in fact, Java can be used to create most other kinds of applications that you would normally use a more conventional programming language to create. HotJava itself is a Java application.

A single Java program can be an applet or an application or both, depending on how you write that program and the capabilities that program uses. Throughout this first week, you'll be writing mostly HotJava applications; then you'll apply what you've learned to write applets in Week 2. If you're eager to get started with applets, be patient. Everything that you learn while you're creating simple Java applications will apply to creating applets, and it's easier to start with the basics before moving onto the hard stuff. You'll be creating plenty of applets in Week 2.

# Creating a Java Application

Let's start by creating a simple Java application: the classic Hello World example that all language books use to begin.

As with all programming languages, your Java source files are created in a plain text editor, or in an editor that can save files in plain ASCII without any formatting characters. On Unix, emacs, ped, or vi will work; on Windows, Notepad or DOS Edit are both text editors.

Fire up your editor of choice, and enter the Java program shown in Listing 1.1. Type this program, as shown, in your text editor. Be careful that all the parentheses, braces, and quotes are there.


**Listing 1.1. Your first Java application.**

```
1: class HelloWorld {
2:     public static void main (String args[]) {
3:         System.out.println("Hello World!");
4:     }
5: }
```

**Warning:** The numbers before each line are part of the listing and not part of the program; they're there so I can refer to specific line numbers when I explain what's going on in the program. Do not include them in your own file.

This program has two main parts:

- ☐ All the program is enclosed in a class definition—here, a class called `HelloWorld`.
- ☐ The body of the program (here, just the one line) is contained in a routine called `main()`. In Java applications, as in a C or C++ program, `main()` is the first routine that is run when the program is executed.

You'll learn more about both these parts of a Java application as the book progresses.

Once you finish typing the program, save the file. Conventionally, Java source files are named the same name as the class they define, with an extension of `.java`. This file should therefore be called `HelloWorld.java`.

Now, let's compile the source file using the Java compiler. In Sun's JDK, the Java compiler is called `javac`.

To compile your Java program, Make sure the javac program is in your execution path and type `javac` followed by the name of your source file:

```
javac HelloWorld.java
```

**Note:** In these examples, and in all the examples throughout this book, we'll be using Sun's Java compiler, part of the JDK. If you have a third-party development environment, check with the documentation for that program to see how to compile your Java programs.

The compiler should compile the file without any errors. If you get errors, go back and make sure that you've typed the program exactly as it appears in Listing 1.1.

When the program compiles without errors, you end up with a file called HelloWorld.class, in the same directory as your source file. This is your Java bytecode file. You can then run that bytecode file using the Java interpreter. In the JDK, the Java interpreter is called simply `java`. Make sure the java program is in your path and type `java` followed by the name of the file without the .class extension:

```
java HelloWorld
```

If your program was typed and compiled correctly, you should get the string `"Hello World!"` printed to your screen as a response.

**Note:** Remember, the Java compiler and the Java interpreter are different things. You use the Java compiler (`javac`) for your Java source files to create .class files, and you use the Java interpreter (`java`)to actually run your class files.

# Creating a Java Applet

Creating applets is different from creating a simple application, because Java applets run and are displayed inside a Web page with other page elements and as such have special rules for how they behave. Because of these special rules for applets in many cases (particularly the simple ones), creating an applet may be more complex than creating an application.

For example, to do a simple Hello World applet, instead of merely being able to print a message, you have to create an applet to make space for your message and then use graphics operations to paint the message to the screen.

**Note:** Actually, if you run the Hello World application as an applet, the `Hello World` message prints to a special window or to a log file, depending on how the browser has screen messages set up. It will not appear on the screen unless you write your applet to put it there.

In the next example, you create that simple Hello World applet, place it inside a Web page, and view the result.

First, you set up an environment so that your Java-capable browser can find your HTML files and your applets. Much of the time, you'll keep your HTML files and your applet code in the same directory. Although this isn't required, it makes it easier to keep track of each element. In this example, you use a directory called HTML that contains all the files you'll need.

```
mkdir HTML
```

Now, open up that text editor and enter Listing 1.2.

### Type Listing 1.2. The Hello World applet.

```
1: import java.awt.Graphics;
2:
3: class HelloWorldApplet extends java.applet.Applet {
4:
5:     public void paint(Graphics g) {
6:        g.drawString("Hello world!", 5, 25);
7:    }
8:}
```

Save that file inside your `HTML` directory. Just like with Java applications, give your file a name that has the same name as the class. In this case, the filename would be `HelloWorldApplet.java`.

Features to note about applets? There are a couple I'd like to point out:

- The `import` line at the top of the file is somewhat analogous to an `#include` statement in C; it enables this applet to interact with the JDK classes for creating applets and for drawing graphics on the screen.
- The `paint()` method displays the content of the applet onto the screen. Here, the string `Hello World` gets drawn. Applets use several standard methods to take the place of `main()`, which include `init()` to initialize the applet, `start()` to start it running, and `paint()` to display it to the screen. You'll learn about all of these in Week 2.

Now, compile the applet just as you did the application, using `javac`, the Java compiler.

```
javac HelloWorldApplet.java
```

Again, just as for applications, you should now have a file called HelloWorldApplet.class in your HTML directory.

To include an applet in a Web page, you refer to that applet in the HTML code for that Web page. Here, you create a very simple HTML file in the HTML directory (see Listing 1.3).

### Type Listing 1.3. The HTML with the applet in it.

```
1: <HTML>
2: <HEAD>
3: <TITLE>Hello to Everyone!</TITLE>
4: </HEAD><BODY>
5: <P>My Java applet says:
6: <APPLET CODE="HelloWorldApplet.class" WIDTH=150 HEIGHT=25>
7: </BODY>
8: </HTML>
```