

## What Is It About Java?

by

Herbert Schildt

Recently, I was asked this question: "What is the one single feature that programmers like most about Java?" Given that Java has many excellent features, this is a non-trivial question! Nevertheless, I have decided to offer my opinion. Before doing so, however, I will begin by answering a slightly different question.

There is something about Java that has caught the imagination of programmers worldwide. This something is not an individual feature, although Java has many good ones, but rather a fundamental aspect of Java's design, something that is woven throughout the fabric of the language. Thus, the question I pose is this: "What is it about Java that makes it resonate so strongly with programmers?"

The answer: it is the expressive power of Java's syntax combined with its streamlined, yet elegant object model. Together they give the programmer the ability to implement sophisticated constructs with ease and reliability. Moreover, these traits lend fluidity to the programming process, enabling the programmer to use the language, rather than fighting it (as is the case with some other computer languages). They combine to make Java a true "programmer's language."

When you add to this foundation the controlled execution environment offered by the Java Virtual Machine (JVM), you have a language that enables *both safe and powerful* code to be written, used, and re-used. This makes Java different from many other languages which are optimized to produce *either* safe code *or* powerful code, but not both. The ability to simultaneously support both safe and powerful code requires a delicate balance, which Java achieves in a seemingly effortless fashion. But, of course, this is the art of Java!

Now, as to what specific Java feature programmers like most, I would suggest garbage collection. In programming languages that do not contain built-in support for garbage collection (such as C++), memory must be managed manually, with the

programmer explicitly releasing unused objects. This is a notorious source of problems because it is easy to forget to release a resource after it is no longer needed, or to release a resource that is still being used. For example, assuming a class called **X**, consider the following C++ function called **doSomething()**.

```
// This is C++ code.
void doSomething() {
    X *obptr = new X;

    if(someError) return; // Trouble here!

    // ...

    delete obptr;
}
```

At the start of **doSomething()**, an **X** object is allocated and a pointer to this object is stored in **obptr**. At the end of **doSomething()**, the object is released via the explicit use of **delete**. In C++, which has no garbage collection, the **delete** statement is necessary because memory must be manually released by the programmer when it is no longer needed. Without **delete**, the memory pointed to by **obptr** would never be freed. At first glance, the function appears correct, with the allocation of memory through **new** balanced by the release of that memory by **delete**. In actuality, however, the function has a serious problem. Notice the **if** statement. If the function returns early due to an error, then the **delete** statement will not be executed. In this case, the memory allocated by **new** is not released. The result is a memory leak.

Java prevents problems such as this by managing memory for you through its garbage collection facility. This can be done in an efficient manner because *all objects* in Java are accessed through a reference. Thus, when the garbage collector finds an object to which there is no reference, it knows that the object is unused and can be recycled. There is no need for the programmer to explicitly release an object. For example, the preceding C++ code can be rewritten in Java as shown here:

```
// Java version.
public void doSomething() {
    X obref = new X();
}
```

```
// ...  
  
if(someError) return; // No problem, now.  
  
// ...  
}
```

In the Java version, memory allocated by the **doSomething( )** method is automatically subject to garbage collection as soon **obref** goes out-of-scope, which occurs when the method returns. Furthermore, it doesn't matter if the method returns early because of an error, or normally. Thus, there is no possibility of a memory leak. It simply can't happen. That is the beauty of Java's garbage collection mechanism.

The prevention of memory management errors is vitally important because, in my experience, they are one of the two leading sources of serious program bugs that are commonly found in commercial software. (The other is the infamous "buffer overrun" error that is presenting such a security risk in much of today's networking software.) By preventing an entire class of difficult-to-find errors, garbage collection lifts a great burden from the Java programmer, freeing him or her to concentrate on creating high-performance software rather than worrying about the minutiae of memory management.

Whether you like Java because of its expressive syntax and streamlined object model, or because of its well-tuned, individual features, we agree that Java is one of the world's preeminent programming languages.

**About Herb Schildt:** Herbert Schildt is a leading authority on the Java, C, C++, and C# languages, and is a master Windows programmer. His programming books have sold more than three million copies worldwide and have been translated into all major foreign languages. He is the author of numerous bestsellers, including *Java 2: The Complete Reference*, *Java 2: A Beginner's Guide*, *Java 2 Programmer's Reference*, *C++: The Complete Reference*, *C: The Complete Reference*, and *C#: The Complete Reference*. Schildt holds a master's degree in computer science from the University of Illinois.