CAMBRIDGE

Cambridge University Press
978-0-521-65892-8 - Java for the COBOL Programmer
E. Reed Doke and Bill C. Hardgrave
Excerpt
More information

Chapter 1

Why You Should Learn Java

Objectives

In this chapter you will study:

- An overview and history of Java
- Some unique characteristics of Java
- Reasons for learning Java
- An overview of the book

So you want to learn Java? Why? You probably want to learn because:

- as a COBOL programmer, you feel a need to update your skills
- Java is hot and so are the jobs for Java programmers
- object technology is hot and Java fits perfectly with the object development your boss is requiring you to learn
- you are a naturally curious person and you want to see what this Java stuff is all about

Whatever your reasons, we're glad you're here!

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-65892-8 - Java for the COBOL Programmer
E. Reed Doke and Bill C. Hardgrave
Excerpt
More information

In this book, the emphasis is on learning Java from a COBOL perspective. There are several million COBOL programmers worldwide and a significant number of you are, or soon will be, learning Java. We have designed this book to help you with that task. We will take what you know about COBOL and apply as much of it as possible to learning Java. But first, let's learn a little bit about Java in general, nontechnical terms, and about object-oriented development. Chapters 1 and 2, respectively, have been set aside for these purposes.

This chapter presents a general description of Java and an overview of the book. Our purpose is to present a brief history of Java, give you an overview of Java, describe why Java is becoming so popular, and discuss Java's impact on the future of COBOL.

Java is a very young language (circa 1995) and its explosive popularity indicates it is rapidly becoming the language of choice for many firms. Three main forces are supporting Java's popularity in information systems:

- the need to increase developer productivity,
- the adoption of the client-server model, and
- the growing number of Internet applications.

Java has received a lot of notice because it can be imbedded in Web pages to do input/output and animation. Java, however, is much more. Java is a complete, industrial-strength, full-featured application development language. It is also object-oriented, which facilitates implementation of the client-server model and Internet applications, plus it can significantly improve your productivity in developing all applications.

On the surface, Java appears to be a totally different language than COBOL. This appearance is because Java is object-oriented and uses a more concise (some say cryptic) syntax than COBOL. Java has numerous similarities with COBOL, however, that can be used to enhance your learning process and ease the transition from COBOL to Java. Learning a new subject (Java) is much easier when you can base it on something you already know (COBOL). That is precisely the basis for this book. We describe Java features and syntax in light of similar features and syntax that exist in COBOL.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-65892-8 - Java for the COBOL Programmer
E. Reed Doke and Bill C. Hardgrave
Excerpt
More information

History and Overview of Java

According to Sun Microsystems, the creator of Java, Java is:

> *"a simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language."*

Enough said? This statement certainly touts many of the benefits of Java, but it doesn't say much about what Java really is. Perhaps we should start with a brief history of Java.

Java began in the early 1990s as a project at Sun Microsystems. The idea was to create a language to support the development of systems embedded in consumer electronic devices. The project, headed by James Gosling, produced a language originally called Oak (named for the tree outside Gosling's window). Unknown at the time, another language had already taken the Oak name. The name was later changed from Oak to Java (the idea for the new name was generated during a visit to a local coffee shop).

C and C++ were available at the time and widely used. However, Sun was concerned about many of the limitations of the popular C++ language. Java is similar to C++, but avoids many of its problems.

After a few years and many changes, the Java language was redirected to the Internet. Fueled by the rapid growth of the Internet and Web applications, Sun officially introduced Java in May 1995. Although Java initially gained recognition and acceptance because of its use on Web pages, it has grown to a full-blown development language capable of producing mission-critical systems on a wide range of hardware and operating systems.

NOTE

1. You may visit Sun Microsystems at *http://java.sun.com/*
2. A glossary of Java and related terms is available at *http://java.sun.com/docs/glossary.print.html*
3. The complete Java Language Specification is available at *http://java.sun.com/docs/books/jls/html/index.html*
4. The Java Language Environment: A White Paper is available at *http://java.sun.com/docs/white/langenv/*

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-65892-8 - Java for the COBOL Programmer
E. Reed Doke and Bill C. Hardgrave
Excerpt
More information

Table 1-1. *Some common Java terms and their meanings*

Applet	A program written in Java that can be embedded inside Web pages and executed in Java-compatible browsers
Bean (i.e., JavaBean)	An independent, reusable software component. Beans can be combined to create an application.
Bytecode	Code generated by a Java compiler. Bytecode is platform independent and must be interpreted by a Java interpreter.
Java Archive files (JAR files)	A Java file format used for compressing many files into one to improve transmission speed across network connections.
Java Development Kit (JDK)	Sun's software development environment for writing Applets and applications in Java.
Java Integrated Development Environment (IDE)	A complete Java development environment including text editors, compilers, debuggers, interpreters, and so forth.
JavaScript	A Web scripting language developed by Netscape; has similar constructs to Java, but is a separate language. Typically used for user interface features within HTML documents, but can also be used on the server side.
Java Virtual Machine (JVM)	A Java interpreter; responsible for interpreting bytecodes.
Servlet	Similar to an Applet, but is located on the server; provides added functionality to Java-enabled servers

The Java name is trademarked by Sun Microsystems, making it a commercial proprietary product. Java is in the public domain, however, and a Java compiler can be obtained from Sun free of charge. Sun has been restrictive in its official licensing of the Java name to assure developers that anything bearing the Java seal meets the language standards and will run reliably in their environment.

During its brief history, Java has created a large following of software developers and a corresponding vocabulary. Many of the commonly used Java terms are listed in Table 1-1.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-65892-8 - Java for the COBOL Programmer
E. Reed Doke and Bill C. Hardgrave
Excerpt
More information

The Popularity of Java

The popularity of Java is evident. Look at the number of Java books available (including this one!). You cannot pick up an industry newspaper or magazine (such as ComputerWorld or InfoWorld) without seeing numerous Java-related items. Java seems to be everywhere—or, at least, everyone is talking about it. In March 1998, the month following the release of version 1.1 of Sun's Java Development Kit, more than 220,000 copies were downloaded from Sun's Web site. The next month, more than 8,000 developers attended the Java One Conference.

But, is Java actually being used? The answer is an emphatic YES! According to Signhal and Nguyen (1998): "Java is being used on a wider range of applications than any other language, including C and C++." In higher education, Java has also gained rapid growth. Java's use in university curricula jumped from 0 percent in 1995 to 42.5 percent in February 1998—not bad for a language officially less than three years old.

Will Java's meteoric growth continue? Only time will tell. Initially, the growth of Java mirrored the growth of the Web, as Java was primarily used as a tool for building Applets. Java's recent growth, however, is in the area of business applications. The popularity of JDK may be a sign of continued growth for Java in business applications.

In its short life, Java has grown faster and gained a wider acceptance than any other language before. Why? What propelled Java, a spin-off of C++, to such success? Certainly, the growth of the Web and Java Applets fueled its growth. However, Java offers many advantages not available in other languages. As an outgrowth of C++, Java specifically avoided some of the problems with C++. Let's take a closer look at how Java differs from other languages.

What Makes Java Different?

Java is just another programming language—right? Yes, it is another programming language, but it is also much more. Some would say the language is simple. For example, Java has only 48 reserved words. At last count, COBOL had over 600! These numbers suggest the language set is much easier to learn and remember than many others, including COBOL—our experience with Java makes us believe this is true.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-65892-8 - Java for the COBOL Programmer
E. Reed Doke and Bill C. Hardgrave
Excerpt
More information

Java is unarguably object-oriented, which puts it in select company (along with other pure object-oriented languages such as Smalltalk and Eiffel). Java is also portable, enabling it to run on many different platforms. Combined, these factors, and more, create an environment of improved development productivity and programming excellence. Let's explore the issues of simplicity, object-orientation, and portability further.

Java Is Simple

Java really is simple, especially if you know C++ because it closely resembles the language. Is it simpler than COBOL? Is it more difficult than Smalltalk? The answers to these questions vary by the individual. It is probably safe to say, however, that Java is easier to learn than C++, but more difficult than BASIC. How's that for a wide range? Our goal in this book is to make Java simple for you, the COBOL programmer. We think you will quickly conclude Java is certainly no more difficult than COBOL to use, and you will come to appreciate and even to prefer its syntax.

Remember the history of Java: Gosling's group at Sun took C++ as the foundation and eliminated many of the negatives. The Eiffel, Smalltalk, and Objective-C languages, among others, also influenced Java. From C++, the troublesome features such as operator overloading, multiple inheritance, and pointers were removed. Features lacking from C++, such as automatic garbage collection and memory management, were added. Overall, Java represents some of the best features and avoids some of the negatives, of several popular languages.

Java Is Object-Oriented

Java is a ***pure*** object-oriented language. *Pure* means everything is represented as an object and all the major characteristics of object-orientation, such as inheritance, classes, and polymorphism are supported (the next chapter explores the characteristics of object-orientation). Smalltalk and Eiffel are examples of other pure OO languages.

In Java, everything is an object except primitive data types (we discuss these in Chapter 4). Single inheritance and polymorphism are supported. A ***hybrid*** language, on the other hand, uses an existing language as its base and adds object-oriented features and syntax. For example,

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-65892-8 - Java for the COBOL Programmer
E. Reed Doke and Bill C. Hardgrave
Excerpt
More information

C++ (derived from C), Object COBOL (derived from COBOL), and Object Pascal (derived from Pascal), are all hybrid languages. In many cases, hybrid languages do not fully support all OO characteristics.

The motivation for hybrid languages is a shorter learning curve. Developers can use their familiar language as the base and simply learn the OO extensions. The danger, of course, is developers do not truly learn OO development. Rather, they learn how to write procedural programs in an OO context. Java forces compliance to object-oriented development. With Java, it is difficult to revert back to procedural programming. However, the use of Java does not ensure a successful development project. Good analysis and design remain essential prerequisites to writing the program code.

In the true spirit of OO, Java applications can be built by assembling JavaBeans in a Java Integrated Development Environment (IDE). In other words, JavaBeans (independent software components or objects) can be combined to form an application. For example, a customer object and bank account object can be combined to form a banking application. Reusable independent software components are the foundation of object-oriented development. Soon, you will write these components using Java and then assemble them into functioning software.

Java Is Portable

Portability is the capability to run an application on different platforms (hardware and software configurations) without making changes to the code. Many of us have experienced the problems associated with moving an application from one platform to another. Typically, it required us to identify code that needed changing and then recompiling and testing every program in the application. Our task would have been trivial if our application had been portable.

With the multitude of platforms available, such as Windows, Windows NT, UNIX, MVS, and OS/2, portability is a most useful feature. Portability is, perhaps, Java's most appealing feature. In fact, "write once, run anywhere" is a common Java slogan.

What gives Java the portability so many other languages lack? Bytecode and Java Virtual Machines (JVM) are key to Java's portability. In most cases, when a program is compiled, machine code is created. For

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-65892-8 - Java for the COBOL Programmer
E. Reed Doke and Bill C. Hardgrave
Excerpt
More information

example, a COBOL program is compiled into machine code that can be executed directly by a computer. Machine code is not portable across various platforms, however, and must be created specifically for the platform to be used. A COBOL program that runs on UNIX probably will not run on MVS and most certainly will not execute on a Windows NT system. In the traditional compiled environment, multiple platforms means multiple compilers, as illustrated in Figure 1-1. If I want my COBOL program to run on UNIX and MVS and Windows NT, I need a compiler for UNIX and MVS and Windows NT. Unfortunately, even with multiple compilers, no assurance exists that the source code will compile and run correctly across all the platforms.

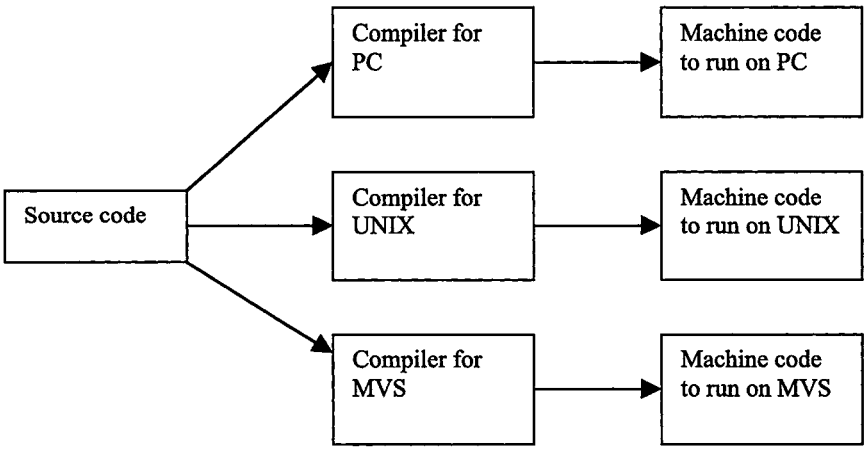


Figure 1-1. *Multiple compilers for multiple platforms*

When Java source code is compiled, it can be compiled into bytecode. Bytecode is not machine code and, therefore, is not directly executable. Instead, bytecode is an intermediate format—somewhere between source code and machine code. For bytecode to be executed on a computer, another tool, called a virtual machine is needed to interpret the bytecodes. A Java Virtual Machine (JVM) is a program that interfaces

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-65892-8 - Java for the COBOL Programmer
E. Reed Doke and Bill C. Hardgrave
Excerpt
More information

with the local operating system and hardware. The JVM interprets the bytecode at runtime so the Java program can run on the local platform, as illustrated in Figure 1-2.

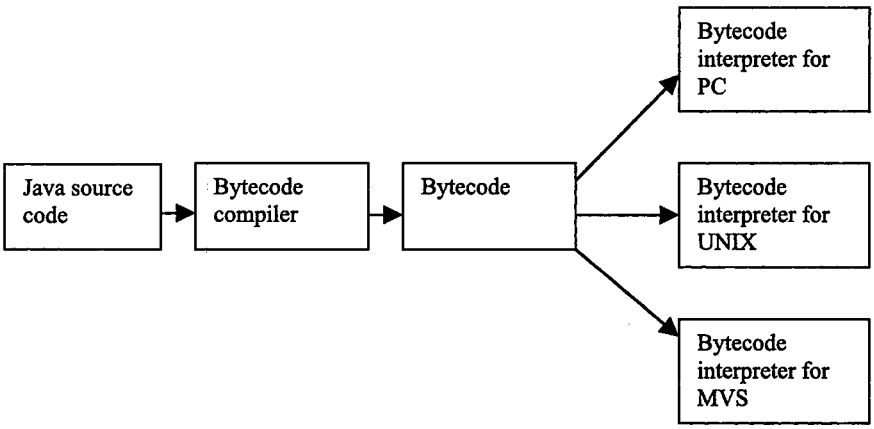


Figure 1-2. *Java running on multiple platforms*

Many different JVMs exist to accommodate the many different platforms. Therefore, only one version of the bytecode file is needed to run on many platforms. Bytecodes are platform independent and portable as long as a JVM is available for the platform for which you want to run the bytecodes. The combination of bytecodes and JVM guarantees portability—no need to re-compile the program for each platform and no need to have multiple compilers for multiple platforms.

Many operating systems fortunately are providing JVMs for their environment. Currently, Windows, Windows NT, Windows CE, Solaris, HP-UX, IRIX, AIW, and MacOS, among others, support Java. And Internet Web browsers such as Netscape and Internet Explorer contain bytecode interpreters, so when a Java Applet is recognized, it is interpreted and executed.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-65892-8 - Java for the COBOL Programmer
E. Reed Doke and Bill C. Hardgrave
Excerpt
More information

As you probably have begun to suspect, however, interpreting a bytecode program is slower than executing native code for a particular platform. In situations where execution speed is an issue, Java can be compiled into native code (just like COBOL) and then it will run as fast as any other application language.

Will Java Replace COBOL?

COBOL's demise has been predicted since its introduction, yet it continues to thrive. Two obvious reasons exist for COBOL's strength. First, it does an incredibly good job of processing and maintaining large quantities of data—the backbone of modern businesses. Second, the language continues to evolve and improve. Each new ANSI standard provides additional features and boosts programmer productivity. The forthcoming COBOL-XX standard provides additional improvements including OO extensions. No, COBOL is not dying. In our view, it isn't even sick!

More than 3 million COBOL programmers are in the world. More than 80 percent of all code, representing as much as 200 billion lines of code, is written in COBOL. Will Java replace COBOL anytime soon? We doubt it. The year 2000 (Y2K) problem has certainly rekindled the demand and interest in COBOL programming, but will this interest continue after January 1, 2000? You may have asked yourself this question. To set the record straight, an increasing demand has occurred for COBOL programmers for many years. Y2K projects may have caused a blip in the demand curve, but the shortage of COBOL people did not begin with Y2K cleanup efforts and will certainly not end with the arrival of the new millennium. Will Java replace COBOL as the primary development language in the long run? Several unanswered questions impact the long run use of COBOL:

- Will OO development continue to be adopted by organizations? The increase, or decrease, in use of OO development in organizations will affect the use of object-oriented languages overall.

- Will Object COBOL be widely accepted? If Object COBOL is widely accepted as the next generation of COBOL, then a smooth migration path is provided to OO development. As such, compa-

© Cambridge University Press www.cambridge.org