December 1996

JAVA: A Research and Educational Overload

John Durrett
University of Texas, Austin

TUTORIAL

JAVA: A RESEARCH AND EDUCATIONAL OVERVIEW

John R. Durrett
University of Texas, Austin

The convergence of computing and telecommunications technologies has created a need for a dynamic, distributed programming language. Sun Microsystems (Sun) hopes that Java will be that language. The goal of this tutorial is to familiarize the audience with the research and educational potential of the Java programming language. It is assumed that the individuals present will not want, and that time constraints forbid, any attempt at a syntax level description of the language. This tutorial will instead attempt to cover the following:

- The basic nature of Java as a programming language;
- Java as it compares to C++;
- Java's strengths and weaknesses; and
- Available development tools and APIs as an indication of industry acceptance and future direction.

1. JAVA

Originally named Oak and renamed Java in early 1995, the Java language is the result of efforts by a small team of Sun employees. It was designed to provide a user interface for control of home appliances (the Green project). When this project failed to produce tangible results and when the extreme growth of the world wide web (WWW) became apparent, the group's efforts shifted to focus the language on a distributed environment and to make it the accepted standard for distributed programming on the Internet. One short year later, the group's efforts are largely successful. Java has been specifically designed to operate as a language for distributed applications on the Internet. The nearest competitors are well behind Java in ability and acceptance. Several factors account for the early success of Java.

Java is architecturally neutral. As a "comterpreted" language, Java is compiled to bytecodes that are designed to run on a virtual computing machine (JVM) rather than to an executable targeted for a single CPU. This method allows consistency across platforms: an integer and a float are always 32 bits, a double always 64 bits, and a short always 16 bits regardless of the system on which Java is running (Flanagan 1996). Programmers get the same API, and executing programs get the same ABI regardless of platform. The fact that Java is architecturally neutral also makes it easier to create student computing facilities. As long as the Java Development Kit is available for the OS that is used (currently SPARC/Solaris, Intel x86/Solaris, Windows NT or 95 and Macintosh 7.5), any platform configuration will do. Java can be taught to students on networked or individual computers, if the networking API is excluded and the file loader is used to run applets.

Java is multithreaded and garbage collected. Multitasking support is built into all levels of the language. At the syntactical level modifiers allow control of individual threads through synchronization. At the object level, a set of class libraries allows the creation of multithreaded programs through inheritance. On machines capable of multitasking, Java's threading capabilities make it a multiprocessor-hot language since threads are mapped to real operating system threads if the operating system supports them. One thread which runs in any Java program is used to release unused memory. This thread frees programmers from the necessity of keeping track of all memory allocated and makes Java programs more efficient in terms of memory leakage. The loss in program speed is moderated by running this thread with a lower CPU priority.

Java is secure. Bytecode verification, lack of direct memory addressing, lack of stack overflows, lack of automatic type conversions and an implicit and easy to understand use of exceptions all make the JVM a secure environment. Untrusted applets execute in a very restricted local environment. All methods and variables are accessed by name rather than by address

to insure valid program flow. Applets loaded from the network are unable to perform any function which impinges on the local environment outside of the applet's restricted operating memory. An applet loaded by the applet loader is prevented from direct memory math, file reads and writes, random network connections and any ability to initiate other programs. In addition, since Java is still a developing language, its creators are able to correct any security holes, such as those found by researchers at Oxford and Princeton, as soon as they are found. Extensions to the basic conservative approach to applet security are under development that would give the JVM the ability to distinguish untrusted applets from trusted ones that have been verified in some way, perhaps through digital signatures.

Java is easy to learn. Its closeness to C and C++ makes it intuitively familiar to many programmers. Many of the features in Java result from its developer's experience with other languages. As examples of such facets, most ordinary statements are taken from C; the idea of interfaces to replace multiple inheritance from Objective C protocols; exceptions as an integral part of the language from Molula-3; dynamic linking and an unused memory collection thread derive from Lisp (Van der Linden 1996). All of these features make it easier for more experienced programmers to understand Java. In addition, the parts of C++ that are not included in Java make the language much easier to learn than C++ since the features that are missing are in many cases the most difficult ones to master.

2. JAVA AS IT COMPARES TO C AND C++

In constructing Java a great effort was made to eliminate the redundancy and the unnecessary complexity from C++. Although some 90% of the constructs in C++ are included in Java, several things are left out.

- There are no more structures. Since everything in Java is an object, there is no requirement for them.
- There are no enumerated types, although the same effect can be created by using a class or an interface to contain only related constants.
- There are no functions; unlike in C++, everything in Java is an object. The basic tenets of object-oriented programming, abstraction, encapsulation, inheritance and polymorphism, are strictly adhered to within the Java language.
- Multiple inheritance and its many problems have been eliminated from Java. Many of the same features can be found in the use of interfaces.
- Operator overloading is missing, but the same effect can be achieved through the correct use of classes.
- Since structures and nonobject arrays are missing from Java and since all objects are accessed by reference, there is no need for direct memory pointers. This simplification alone eliminates the primary source of program bugs, intentional and not, from Java code.
- There is no automatic coercion; all type conversions in which precision is lost must be explicit. There are also no implicit declarations in Java. This facet enables much more stringent compile and run time error checking.
- Bounds checking is added to Java to prevent memory overruns and to restrict use of memory to valid addresses.
- The elimination of all preprocessor statements is intended to make Java simpler that C++. There are no typedefs, defines, or header files. This simplification makes it much easier to read and to understand a Java program. There is no need to understand a program's context from other files; each Java file is much more self-explanatory.

3. LANGUAGE WEAKNESSES

Java is intended to be a distributed, flexible, dynamic language, and these features are not without cost. One of the primary problems with Java is that it is very slow, compared to almost any compiled language, and about twenty times slower than C (Van Hoff, Shaio and Starbuck 1996). This sluggishness is unsurprising since the bytecode must be verified and executed line-by-line and is not optimized for a specific platform. Just in time compilers, such as the ones from Borland and Symantec, efficient bytecodes, and multithreading all help alleviate this problem. Another problem with Java is its lack of dynamic mobility. There is no provision made for process migration or local persistence. Java applets are typically downloaded by a client, and security restrictions prevent dynamic movement or client-side file creation. Once a facility for execution of trusted applets is provided, many of these issues may be alleviated. Another solution to this problem is to use another language. General Magic is working on using Telescript to provide the mobility and Java to provide the client-side content. Sun is also working on the problem using TCL to provide the required mobility. A final problem is that there is no provision made in the language for a uniform communication model. Any applet-to-applet communications are strictly up to the developers; no cross-developer standardization exists at a level higher than that of the Corba standards. This problem can possibly be eliminated once agent researchers find an acceptable software agent communication standard, such as KQML.

4. INDUSTRY ACCEPTANCE: FAD OR FUNDAMENTAL

Sun's implementation of a language that is secure, portable, and dynamic and intended as a distributed computing platform seems to have been successful. It is difficult to open an information systems magazine and not find an indication that Java is here to stay. For example, OpenDoc is being extended to embrace the Java environment; IBM (both OS2 and System/390 divisions), Apple, Lynx, and Microsoft are all extending their operating systems to include support for Java; and Novell is planning to rewrite many of its major NetWare utilities in Java to enable quicker ports to other platforms. While competition from other languages, such as Microsoft's ActiveX and Borland's Delphi Visual Component libraries, could be seen as potential problems for Java, none offer the level of portability nor focus as clearly on the realm of distributed computing as Java does. All indications point to the continuing progress of the Java juggernaut (Guadagno and Okon 1996) across the development world.

5. REFERENCES

Flanagan, D. *Java in a Nutshell*. Sebastopol, California: O'Reilly & Associates, Inc., 1996.

Gosling, J., and McGilton, H. "The Java Language Environment: A White Paper." Mountain View, California: Sun Microsystems, 1996. *http://java.sun.com:80/doc/language_environment/CreditsPage.doc.html#Contents*.

Guadagno, L., and Okon, O. B. "Stepping Up the Flow of Business Information." *Object Magazine*, Volume 6, July 1996, pp. 41-45.

Van der Linden, P. *just Java*. Mountain View, California. Sunsoft Press, 1996.

Van Hoff, A.; Shaio, S.; and Starbuck, O. *Hooked on Java*. New York: Addison-Wesley, 1995

All examples to be used in this tutorial can be found at: *http://www.bus.utexas.edu/~durrett/ICIS/*