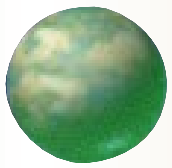
111000111000000111100000011111110000101010
0101010101010101010101010100000111110101010

Sandeep Singhal and Binh Nguyen

THE JAVA FACTOR

The "write once, run anywhere" slogan is synonymous with the Java programming language. The Java run-time library provides application developers with the ability to write code in one single language and the confidence that the code will execute without modification on virtually any hardware, any operating system, and any application environment. Developers are freed from the costs of porting their applications to a myriad of target platforms and worse, of maintaining those multiple-code bases. Of course, powerful cross-platform run-time systems have existed for many years, dating back to Lisp, Smalltalk, and many other interpreted languages. However, Java is the first to achieve widespread popularity among commercial developers.

When Sun Microsystems introduced Java in 1995, the language was primarily used for developing applets—downloadable mini-applications that could be embedded inside Web pages and executed in browsers. If cross-platform portability was the only advantage of Java, the language probably would have gone no further. However, in the three years since its commercial introduction, Java has emerged as a first-class programming language that is being used for everything from embedded devices to enterprise servers. The language today is seeing use in a wider range of applications than any other language, including C and C++.



Java Language and Platform Attraction

The Java programming language [4] is a strongly typed, object-oriented language that borrows heavily from the syntax of C++ while avoiding many of the C++ language features that lead to programming errors, obfuscated code, or procedural programming. Removed from C++ are pointers and pointer arithmetic, operator overloading, struct and union, and multiple inheritance. All public Java functions are virtual (and private virtual functions are not supported), and the goto statement is gone. Its strong typing means arbitrary type conversions are disallowed. Instead of requiring the application to manage its heap-allocated memory, the Java run-time environment provides automatic garbage collection, enabling application developers to minimize the dual dangers of memory leaks and erroneous memory references.

Java is designed to directly support OO programming. All data and functions must be encapsulated within classes, much like Smalltalk. Interfaces are first-class language constructs and are enforced by the type system. A class may be declared to implement multiple interfaces simultaneously, though it may only inherit from one implementation class. The Java run time includes class libraries that provide high-level interfaces for GUI programming, I/O, multithreading, and networking [6].

Using Java, one can restrict the set of resources (files, threads, network, and so forth) that applications may access on the local machine. Though originally intended to support the execution of untrusted applets downloaded from the Web, the need for security has proven to be important in a broader context. The next version of Java [7] will support security restrictions on a per-class basis, allowing the simultaneous execution

MATSU

The Java language is rapidly becoming an integral part of the enterprise environment, where it provides the glue between transaction monitors, persistent storage, and concurrency, object life cycle, and naming services.

of "trusted" and "untrusted" applications or the deployment of applications that dynamically load "add-on" components from untrusted third parties.

In all, Java exposes a relatively simple, powerful and elegant programming model. It is object oriented and therefore conforms naturally to the established design patterns [3] used throughout the standard class libraries. In this section, James Cooper discusses how the Java language facilitates elegant OO design choices.

Java source code is compiled into bytecodes that are interpreted at run time by a virtual machine (VM) [11]. These bytecodes give Java its portability; as long as a virtual machine exists for the target platform, the Java application should be able to execute without intervention. Classes are loaded into the virtual machine on demand either from the local file system or over the network from a Web server. As the code is loaded into the VM, a verifier ensures the bytecodes are valid and properly constructed. As the bytecodes are executed, a just-in-time (JIT) compiler can perform dynamic optimizations, including converting the bytecodes into native machine instructions for faster execution.

Widespread adoption of Java is driving rapid improvements in Java VM performance and the performance of JIT compilers. The ubiquity and power of the Java run-time platform have made it into an attractive target platform for a variety of other programming languages. For example, the NetRexx language [2], a scripting language intended for non-programmers, is compiled into Java bytecodes and can therefore be executed in the standard Java run-time environment.

Application Environments Big and Small

People are probably most familiar with Java's traditional uses, namely Web applets and desktop applications. However, these uses for Java are evolving rapidly from the simple data entry and user scripting tasks of the past. The new Java Media APIs [8] support 2D and 3D graphics, playback and recording of multimedia data, speech recognition and synthesis, and even telephony. Together, these APIs enable the development of powerful Java applications rivaling those otherwise developed using native operating system functions.

Java is even adopting new roles over the Internet. Integrated with the Virtual Reality Modeling Language (VRML) for describing interactive 3D scenes, Java allows the introduction of dynamic behavior models for VRML objects and multiuser interactions in these virtual worlds. In this section, Don Brutzman describes VRML and how it leverages the Java language. To enable a new generation of networked applications, new Java toolkits support the interactive sharing of data and events by multiple users. The Java Shared Data Toolkit (JSDT) [9] from JavaSoft and Shared Data Objects (SDO) [5] from IBM are examples of these new application environments.

At one extreme, Java is being used in embedded devices for smart cards and digital cash, telephones, television set-top boxes, and household appliances. For Java, these applications actually represent a return to Java's roots, for James Gosling's research group at Sun originally designed the language specifically for these environments. Scaled-back versions of the Java class libraries, Personal-Java, Embedded-Java, and JavaCard [10], execute with minimal memory requirements. Using the portable Java run time, devices can provide encryption and authentication services, as well as basic network communications and data exchange. For devices with a display screen, the application developer can implement a basic GUI, including a Web browser.

When running on embedded devices, current Java applications lack the ability to explicitly manage how much memory and CPU time are given to each executing task. Moreover, the application cannot assess what system resources are available to determine whether a particular task should execute. In his article, Kelvin Nilsen explores how the Java environment can be augmented to address the issues facing real-time Java applications.

The Java language is rapidly becoming an integral part of the enterprise environment, where it provides the glue between transaction monitors, persistent storage, and concurrency, object life cycle, and naming services. Using the Enterprise JavaBeans (EJB) component model [1, 12], one might purchase a data storage container (such as a database) and transaction monitor (supporting concurrency, distribution, replication, and so forth) from one vendor; one might purchase business logic (such as a system for managing bank accounts) from another vendor. After customizing the business logic and integrating other independently developed software components, the application developer uses EJB tools to link the business logic to the container.

Most important, the application logic can be ported to use a different container without modification.

Enterprise computing stands to benefit considerably from the use of Java. For example, portability across a range of persistence services means application can be migrated from an entry-level transaction system to a high-throughput distributed transaction system as requirements change. Persistent storage can be re-targeted between a flat file, a relational database, or an object database based on price-performance demands. Enterprise Java has support from all the major systems vendors including Oracle, Sybase, and IBM.

An integral part of the Java enterprise computing model is the specialized business logic that supports particular types of applications. For example, in their article, Kathy Bohrer, Verlyn Johnson, Anders Nilsson, and Bradley Rubin describe a general distributed business object framework that supports a variety of common business objects such as Currency, Business Partner, and Address. This framework is being customized to support particular applications, including financial account management, order processing, and warehouse/inventory management.

Interestingly enough, this emergence of enterprise computing brings Java back to its Web roots. Modern Web servers (better referred to as application servers) provide support for servlets—Java applications that are invoked in response to HTTP requests. Servlets provide Web users with an entry point for accessing and manipulating a wide-range of enterprise data. As Java code, servlets can use Java Database Connectivity (JDBC) to access SQL databases, Java Naming and Directory Interface (JNDI) to access LDAP and other corporate directory information, and EJBs to access transaction and persistence services. Ultimately, Java is serving to fulfill the vision of providing ubiquitous access to information.

Getting Started

Now is an exciting time for Java. The language promises to touch our lives in numerous ways, whether it be in our ATM and debit cards, our telephones, our household appliances, our desktop computers, or the payroll and ordering systems used by our employers. For this reason, and because of its overall simplicity and relative purity as an OO language, Java is rapidly becoming the subject of first-year programming courses. Paul Tyma summarizes the arguments for and against Java in the first article of the section.

Fortunately, there are plenty of resources available to learn more about Java. Please see the accompanying box for some pointers. C

References

1. Copeland, G. S. Claussen, Conner, M. and Hambrick, G. *Enterprise Programming with Java*. Oct. 1997.
2. Cowlishaw, M. NetRexx-Easier Java Programming. *Java Report* 2, 6 (June 1997), 45–48.
3. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading Mass., 1995.
4. Gosling, J. Joy, B. and Steele, G. *The Java Language Specification* Addison-Wesley, Reading Mass., 1996.
5. International Business Machines, Corp. Shared Data Objects; www.alphaWorks.ibm.com/.
6. JavaSoft, Inc. Java Development Kit 1.1.5; www.javasoft.com/jdk/1.1/.
7. JavaSoft, Inc. Java Development Kit 1.2; www.javasoft.com/jdk/1.2/.
8. JavaSoft, Inc. Java Media; java.sun.com/products/java-media/.
9. JavaSoft, Inc. Java Shared Data Toolkit; developer.javasoft.com/developer/earlyAccess/jsdt/.
10. JavaSoft, Inc. Products and APIs; java.sun.com/products.
11. Lindholm, T. and Yellin, F. *The Java Virtual Machine Specification.* Addison-Wesley, Reading, Mass., 1997.
12. Thomas, A. Enterprise JavaBeans: Server component model for Java; java.sun.com/products/ejb/white_paper.html.

Sandeep Singal (singhal@us.ibm.com) is a research staff member in the IBM T.J Watson Research Center in Hawthorne, NY.
Binh Nguyen (binhn@us.ibm.com) is a senior software engineer in the IBM Software Solutions Division in Research Triangle Park, NC.

POINTERS BOX

Books

P. Niemeyer, J. Peck, *Exploring Java*, Second Edition, O'Reilly, 1997.

C. Horstmann and G. Comell, *Core Java 1.1, Volume I: Fundamentals,* Prentice-Hall, 1997.

C. Horstmann and G. Comell, *Core Java 1.1, Volume II: Advanced Features,* Prentice-Hall, 1997.

D. Berg and S. Fritzinger, *Advanced Techniques for Java Developers,* Wiley, 1998.

Magazines

Java Report; www.sigs. com/jro.
Java World; www.javaworld.com/.
Pure Java Developer's Journal; www.cobb.com/pjd/.

Usenet

See the comp.lang.java newsgroup hierarchy as well as comp.compilers.tools.javacc.