Java: Object-oriented programming for the cyber age

SIAMAK HASSANZADEH
Sun Microsystems
Mountain View, California

CHARLES C. MOSHER
ARCO Exploration and Production Technology
Plano, Texas

A phenomenal rise in interest in the Java programming environment and World Wide Web browsers with Java capability have added a new dimension to the ever-increasing popularity of the Internet and the Web *(TLE,* March and July 1995).

The Java programming language originated at Sun Microsystems as part of a larger project to develop advanced software for consumer electronics. These devices are small, reliable, portable, distributed, real-time embedded systems. Initially the intention was to use C++. But after encountering a number of problems, a new language, originally called Oak, was born. After several years of experience with the language, it was renamed to Java and was retargeted to the Internet. In a nutshell, Java is a simplified, safe, and portable version of C++. Examples of Java aware browsers are Netscape (2.0 and above) and the HotJava Browser from Sun Microsystems. The HotJava browser is actually an application written in the Java language. Web browsers with Java capabilities help make the Internet come alive. Java-based browsers build on the Internet and network browsing techniques established by Mosaic *(TLE,* March 1995) and expands them by adding dynamic behavior that transforms static documents into dynamic applications.

One of the most useful features of Java is its ability to dynamically add to its capabilities. This feature is called interactive content. Using Java, you can add applications that allow the user to interact dynamically with the Web page. Examples in geophysics range from interactive seismic data interpretation to 3-D seismic movies (dynamic visualization of 3-D seismic data); the possibilities are endless.

Design goals. The Java programming language and environment are designed to solve a number of problems in modern programming practice. Java is a "simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic" language. (The Java language: http://java.sun.com.)

From the onset of its design, Java was meant to be simple and familiar. It has the "look and feel" of C and C++, and as such it can be programmed easily and without much training. The objective from the beginning was that the learning curve should not be too steep.

Java is object-oriented, providing facilities to define and manipulate objects. Objects are self-contained entities which have a state (a collection of data associated with an object) and to which messages can be sent (see Appendix A for a brief introduction to objects). A message is a mechanism by which the state of an object can be accessed or altered. The object-oriented facilities of Java are essentially those of C++, with extensions from Objective C.

Designed for Intranet and Internet computing, Java has an extensive library of routines for dealing with TCP/IP protocols like HTTP and FTP *(TLE,* March and July 1995). Network computing is integrated into the language and runtime system and is hence (almost) transparent. Moreover, a great deal of emphasis has been placed on security. Java provides a programming environment for developing virus-free and tamper-free applications. Applications written in Java are secure from intrusions by unauthorized codes. It should be noted, however, that not all security issues of the Java language have been resolved. These issues are being addressed as the language evolves.

Java is robust due to the elimination of features of programming languages such as C which lead to applications crashing at crucial moments. The single biggest difference between Java and C/C++ is that Java has a pointer model that eliminates the possibility of overwriting the memory and corrupting data. Instead of pointer arithmetic, Java has true arrays as in Fortran. In addition, it's not possible to turn an arbitrary integer into a pointer by casting. Java puts a lot of emphasis on early checking for possible problems, later dynamic (runtime) checking, and eliminating situations that are error prone.

Support applications of Java run on heterogeneous networked computer systems. To enable a Java application to execute anywhere on the network, the compiler generates *bytecodes* - an architecture neutral object file format designed to transport codes to multiple hardware and software platforms. In one sense, bytecodes are portable hardware-level machine instructions. The Java *interpreter* can execute bytecodes directly on any system to which the interpreter has been ported. With this neutrality, most applications software written in the Java language can be run on any system. The architecture-neutral and portability of Java is often referred to conceptually as the *Java Platform,* a "Write Once, Run Anywhere" capability for delivering and running applications on heterogeneous computing environments.

High performance is achieved by translating bytecodes on the fly (at runtime) into native machine codes. For applications requiring large amounts of compute power, the compute-intensive segments can be rewritten in native machine codes and interfaced with the Java environment. Java also supports multithreading; the capability to execute multiple concurrent sequences of instructions. Multithreading allows development of high performance parallel computing applications. The inclusion of parallel computing concepts as part of the language makes Java an attractive platform for not only for interactive applications, but also for scientific computing applications (i.e., seismic processing) requiring large amounts of computing horsepower.

A more dynamic language than C or C++, Java can adapt to an evolving environment. Many functionalities needed during the execution of an application can be linked dynamically as needed. In summary, the Java language makes object-oriented programming easier and

```
*The HelloWorldApp class implements an application that
*simply displays" Hello World!' to the standard output.
*/
class HelloWorldApp {
    public static void main(String[] args) {
       System.out.println("Hello World!"); //Display the string.
```

Figure 1. The 'Hello World!' program written in Java.

```
int sum (int arr[]) {
    int result = 0;
    for (int i = arr.length ; --i>=rl ;) {
          results += arr{i};
    }
    return results;
}
```

Figure 2. The 'Sum' program written in Java.

```
//Iterative
            long faci (int n) (
                long result = 1 L;
                for (int i = 2; i n ; i++ ) {
                      results*= n:
                }
                return  results;
            }
// Recursive
            long facr (int n) {
                if(n<=I){
                return 1 L;
                }
                retturn facr (n - 1) * n;
            }
```

Figure 3. The 'Factorial' program written in Java.

provides a new powerful tool for the development of interactive, distributed, and parallel computing applications.

Language specification. Although related to C and C++, Java is organized differently, with some of their features omitted and a few from other languages included. Java is intended to be a production language, not a research language, and so the design of Java has avoided including new and untested features.

Java is strongly typed language, which means that every variable and every expression has a type that is known at compile time. Types limit the values that a variable can hold or that an expression can produce. Types also limit the operations supported on those values, and determine the meaning of the operations. This specification clearly distinguishes between errors that can and must be detected at compile time, and those that occur at run time. Compile time normally consists of translating Java programs into a machine-independent byte-code representation. Run-time activities include loading and linking of the classes needed to execute a program, optional machine code generation and dynamic optimization of the program, and actual program execution.

As in C or C++, Java is a relatively high-level language. To illustrate the simple and familiar features of Java, Figure I displays the traditional 'Hello World!' program written in Java. This example declares a class thus named. Within this class, a single method called 'main' is declared which in turn contains a single method invocation to display the string 'Hello World!' on the standard output (e.g., on the monitor). The statement that prints 'Hello World!' does so by invoking the 'println' method of the 'out' object. The out object is a class variable in the 'system' class that performs output operations on tiles. Figures 2 and 3 show examples of a summing operation and factorial written in Java language. More detailed information and examples can be obtained from http:// java.sun.com.

JavaSeis. Seismic data processing is ideally suited for parallel distributed computing *(TLE,* this issue). However, thus far there has not been a simple programming environment to provide a portable, distributed and parallel framework for scientific computation in general, and seismic data processing in particular. Previous attempts at developing such a framework have been based on C++ (POOMA: the parallel object-oriented methods and applications, http://www.acl.lanl.gov/pooma-framework).

As the basis for the JavaSeis efforts, we plan to use the ARCO seismic benchmark suite (SBS), which has recently been incorporated into the SPEChpc96 benchmark suite as SPECseis96. SBS manages seismic data as parallel distributed objects using traditional procedural languages (C and Fortran). The environment is designed to be flexible and extendable, and is used extensively at ARCO as a prototyping en-

00 programming 101

Object-oriented programming (in contrast to procedural techniques) is a different way of organizing software. *Objects* are software programming entities that may represent real world items such as cars, planes, trees, oil fields, etc. Software applications may also contain objects such as spreadsheets, menus, and so on. Thus any system can be regarded as a collection of objects.

In technical terms, object-oriented technology consists of a set of nodules called classes, a set of inheritance relations, and mapping 'unctions that associate pairs of classes. Each object is represented by a unique identifier and has some data associated with it. The collecion of data associated with an obect is called its *state.* The state of an object is defined by its *instance variables.* The functions an object performs are known as its *behavior* An object's behavior is defined by *methods.* Methods manipulate the instance variables to create a new state. A class is a software construct hat defines the instance variables and methods of an object. A class in and of itself is not an object. A class defines how an object will look and behave when an object is created or nstantiated from the specification declared by the class. *Inheritance is* a defined relationship between two classes

If an object wants another object to perform some work on its behalf it sends a message to the second ob. ject. Using the message-passing par. adigms of object-oriented pro. gramming, one can build entire net. works of objects that pass message! among themselves to change their states. This programming technique is ideal for creating a distributed and parallel framework for scientific computing. Appendix B contains a list of references on object-orientec programming.

vironment. New processes are developed using "inheritance by copying templates." As a result, system changes that often require extensive modifications to all copies of a few basic system templates are now amended by adjusting only base templates. Parallelism is managed through an API defined by the services required for implementing common geophysical data processing algorithms .

The existing object-based structure of SBS and the well-defined parallel behavior of many geophysical algorithms makes this application suite a natural fit with the constructs defined in the Java language. Currently, maintenance and portability of SBS are made more difficult by lack of a coherent object model and a consistent parallel programming model. Implementation of SBS in Java would significantly improve the portability and maintainability of seismic computing applications, and would also provide a pathway to use of network based parallel computing for production seismic processing.

Concluding remarks. The Java language, very simple and object-oriented, provides an extremely attractive platform for development of portable, distributed and parallel applications for scientific computing in general and seismic data processing in particular. Java's architecture-neutral aspects make it an ideal development programming paradigm to meet the challenges of distributing dynamically extensible software across networks of computers.

Suggestions for further reading. *An introduction to Object-Oriented Programming*, by T. Budd., Addison Wesley Publishing Co. includes a comparison of the C++, Objective C, SmallTalk, and Object Pascal. The *Design and Evolution of C++*, by B. Stroustrop, Addison Wesley , is a detailed history of how we came to where we are with *C++*. *Java Language Tutorial: Object-Oriented Programming for the Internet*, Sun Microsystems, Inc. http://www.javasoft.com/ is an on-line guide to writing programs in the Java language. *The Java Language Specification*, by Gosling, Joy, and Steele, Sun Microsystems, http://www.javasoft.com/. Another introduction to the object oriented techniques is *Active Java, Object-oriented Programming for the World Wide Web*, by Freeman and Ince, Addison-Wesley. TLE