Java: an Overview

James Gosling, February 1995
jag@sun.com

Introduction

JAVA† is a programming language and environment that was designed to solve a number of problems in modern programming practice. It started as a part of a larger project to develop advanced software for consumer electronics. These are small reliable portable distributed real-time embedded systems. When we started the project, we intended to use C++, but we encountered a number of problems. Initially these were just compiler technology problems, but as time passed we encountered a set of problems that were best solved by changing the language.

This document contains a lot of technical words and acronyms that may be unfamiliar. You may want to look at the glossary on page 8.

There is a companion paper to this, *WEBRUNNER an Overview*, that describes a very powerful application that exploits JAVA.

JAVA

JAVA: A simple, object oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high performance, multithreaded, dynamic language.

One way to characterize a system is with a set of buzzwords. We use a standard set of them in describing JAVA. The rest of this section is an explanation of what we mean by those buzzwords and the problems that we were trying to solve.

> *Archimedes Inc. is a fictitious software company that produces software to teach about basic physics.This software is designed to interact with the user, providing not only text and illustrations in the manner of a traditional textbook, but also providing a set of software lab benches on which experiments can be set up and their behavior simulated. For example, the most basic one allows students to put together levers and pulleys and see how they act. A narrative of their trials and tribulations is used to provide examples of the concepts presented.*

Simple

We wanted to build a system that could be programmed easily without a lot of esoteric training and which leveraged today's standard practice. Most programmers working these days use C, and most doing object-oriented

† The internal development name for this project was "Oak" . JAVA is the new official product name..

programming use C++. So even though we found that C++ was unsuitable, we tried to stick as close as possible to C++ in order to make the system more comprehensible.

JAVA omits many rarely used, poorly understood, confusing features of C++ that in our experience bring more grief than benefit. This primarily consists of operator overloading (although it does have method overloading), multiple inheritance, and extensive automatic coercions.

Paradoxically, we were able to simplify the programming task by making the system somewhat more complicated. A good example of a common source of complexity in many C and C++ applications is storage management: the allocation and freeing of memory. JAVA does automatic garbage collection —this not only makes the programming task easier, it also dramatically cuts down on bugs.

> *The folks at Archimedes wanted to spend their time thinking about levers and pulleys, but instead spent a lot of time on mundane programming tasks. Their central expertise was teaching, not programming. One of the most complicated of these programming tasks was figuring out where memory was being wasted across their 20K lines of code.*

Another aspect of simple is small. One of the goals of JAVA is to enable the construction of software that can run stand-alone in small machines. The size of the basic interpreter and class support is about 30K bytes, adding the basic standard libraries and thread support (essentially a self-contained microkernel) brings it up to about 120K.

Object-Oriented

This is, unfortunately, one of the most overused buzzwords in the industry. But object-oriented design is still very powerful since it facilitates the clean definition of interfaces and makes it possible to provide reusable “software ICs”.

A simple definition of object oriented design is that it is a technique that focuses design on the data (=objects) and on the interfaces to it. To make an analogy with carpentry, an “object oriented” carpenter would be mostly concerned with the chair he was building, and secondarily with the tools used to make it; a “non-OO” carpenter would think primarily of his tools. This is also the mechanism for defining how modules “plug&play”.

The object-oriented facilities of JAVA are essentially those of C++, with extensions for more dynamic method resolution that came from Objective C.

> *The folks at Archimedes had lots of kinds of things in their simulation. Among them, ropes and elastic bands. In their initial C version of the product, they ended up with a pretty big system because they had to write separate software for describing ropes versus elastic bands. When they re-wrote their application in an object oriented style, they found they could define one basic object that represented the common aspects of ropes and elastic bands, and then ropes and elastic bands were defined as variations (subclasses) of the basic type. When it came time to add chains, it was a*

snap because they could build on what had been written before, rather than writing a whole new object simulation.

Distributed

At one time networking was integrated into the language and runtime system and was (almost) transparent. Objects could be remote: when an application had a pointer to an object, that object could exist on the same machine, or some other machine on the network. Method invocations on remote objects were turned into RPCs (Remote Procedure Calls).

A distributed application looked very much like a non-distributed one. Both cases used an essentially similar programming model. The distributed case did, however, require that applications paid some attention to the consequences of network failures. The system dealt with much of it automatically, but some of it did need to be dealt with on a case-by-case basis.

Since then, that model has mostly disappeared: a concession to the pragmatics of living within the context of existing networks. Primarily the Internet. Consequently, JAVA now has a very extensive library of routines for easily coping with TCP/IP protocols like http and ftp. JAVA applications can open and access objects across the net via URLs with the same ease the programmers are used to accessing a local file system.

The folks at Archimedes initially built their stuff for CD ROM. But they had some ideas for interactive learning games that they'd like to try out for their next product. For example, they wanted to allow students on different computers to cooperate in building a machine to be simulated. But all the networking systems they'd seen were complicated and required esoteric software specialists. So they gave up.

Robust

JAVA is intended for writing programs that need to be reliable in a variety of ways. There is a lot of emphasis on early checking for possible problems, later dynamic (runtime) checking, and on eliminating situations which are error prone.

One of the advantages of a strongly typed language (like C++) is that it allows extensive compile-time checking so bugs can be found early. Unfortunately, C++ inherits a number of loopholes in this checking from C, which was relatively lax (the major issue is method/procedure declarations). In JAVA, we require declarations and do not support C style implicit declarations.

The linker understands the type system and repeats many of the type checks done by the compiler to guard against version mismatch problems.

A mentioned before, automatic garbage collection avoids storage allocation bugs.

The single biggest difference between JAVA and C/C++ is that JAVA has a pointer model that eliminates the possibility of overwriting memory and corrupting data. Rather than having pointer arithmetic, JAVA has true arrays. This allows subscript checking to be performed. And it is not possible to turn an arbitrary integer into a pointer by casting.

> *The folks at Archimedes had their application basically working in C pretty quickly. But their schedule kept slipping because of all the small bugs that kept slipping through. They had lots of trouble with memory corruption, versions out-of-sync and interface mismatches. What they gained because C let them pull strange tricks in their code, they paid for in Quality Assurance time. They also had to reissue their software after the first release because of all the bugs that slipped through.*

While JAVA doesn't pretend to make the QA problem go away, it does make it significantly easier.

Very dynamic languages like Lisp, TCL and Smalltalk are often used for prototyping. One of the reasons for their success at this is that they are very robust: you don't have to worry about freeing or corrupting memory. Programmers can be relatively fearless about dealing with memory because they don't have to worry about it getting messed up. JAVA has this property and it has been found to be very liberating. Another reason given for these languages being good for prototyping is that they don't require you to pin down decisions early on. JAVA has exactly the opposite property: it forces you to make choices explicitly. Along with these choices come a lot of assistance: you can write method invocations and if you get something wrong, you get told about it early, without waiting until you're deep into executing the program. You can also get a lot of flexibility by using interfaces instead of classes.

Secure

JAVA is intended to be used in networked / distributed situations. Toward that end a lot of emphasis has been placed on security. JAVA enables the construction of virus-free, tamper-free systems. The authentication techniques are based on public-key encryption.

There is a strong interplay between "robust" and "secure". For example, the changes to the semantics of pointers make it impossible for applications to forge access to data structures or to access private data in objects that they do have access to. This closes the door on most activities of viruses.

Not included in release 0.1 or 0.2.

There is a mechanism for defining approval seals for software modules and interface access. For example, it is possible for a system built on JAVA to say "only software with a certain seal of approval is allowed to be loaded" and it is possible for individual modules to say "only software with a certain seal of approval is allowed to access my interface". These approval seals cannot be forged since they are based on public-key encryption.

> *Someone wrote an interesting "patch" to the PC version of the Archimedes system. They posted this patch to one of the major bulletin boards. Since it was easily available and added some interesting features to the system, lots of people downloaded it. It hadn't been checked out by the folks at Archimedes, but it seemed to work. Until the next April first when thousands of folks discovered rude pictures popping up in their children's lessons. Needless to say, even though they were in no way responsible for the incident, the folks at Archimedes still had a lot of damage to control.*

Architecture Neutral

JAVA was designed to support applications on networks. In general, networks are composed of a variety of systems with a variety of CPU and operating system architectures. In order for an JAVA application to be able to execute anywhere on the network, the compiler generates an architecture neutral object file format — the compiled code is executable on many processors, given the presence of the JAVA runtime.

This is useful not only for networks but also for single system software distribution. In the present personal computer market, application writers have to produce versions of their application that are compatible with the IBM PC and with the Apple Macintosh. With the PC market (through Windows / NT) diversifying into many CPU architectures, and Apple moving off the 68000 towards the PowerPC, this makes the production of software that runs on all platforms almost impossible. With JAVA, the same version of the application runs on all platforms.

The JAVA compiler does this by generating bytecode instructions which have nothing to do with a particular computer architecture. Rather, they are designed to be both easy to interpret on any machine and easily translated into native machine code on the fly.

Archimedes is a small company. They started out producing their software for the PC since that was the largest market. After a while, they were a large enough company that they could afford to do a port to the Macintosh, but it was a pretty big effort and didn't really pay off. They couldn't afford to port to the PowerPC Mac or MIPS NT machine. They couldn't "catch the new wave" as it was happening, and a competitor jumped in...

Portable

Being architecture neutral is a big chunk of being portable, but there's more to it than just that. Unlike C and C++ there are no "implementation dependent" aspects of the specification. The sizes of the primitive data types are specified, as is the behaviour of arithmetic on them. For example, "int" always means a signed twos complement 32 bit integer, and "float" always means a 32 bit IEEE 754 floating point number. Making these choices is feasable in this day and age because essentially all interesting CPUs share these characteristics.

The libraries that are a part of the system define portable interfaces. For example, there is an abstract Window class and implementations of it for Unix, Windows and the Mac†.

The JAVA system itself is quite portable. The new compiler is written in JAVA and the runtime is written in ANSI C with a clean portability boundary. The portability boundary is essentially POSIX.

Interpreted

The JAVA interpreter can execute JAVA bytecodes directly on any machine to which the interpreter has been ported. And since linking is a more incremental

† The Windows and Mac versions aren't complete yet.

& lightweight process, the development process can be much more rapid and exploratory.

As a part of the bytecode stream, more compile-time information is carried over and available at runtime. This is what the linker's type checks are based on, and what the RPC protocol derivation is based on. It also makes programs more amenable to debugging.

> *The programmers at Archimedes spent a lot of time waiting for programs to compile and link. They also spent a lot of time tracking down senseless bugs because some changed source files didn't get compiled (despite using a fancy "make" facility), which caused version mismatches; and they had to track down procedures that were declared inconsistently in various parts of their programs. Another couple of months lost in the schedule.*

High Performance

While the performance of interpreted bytecodes is usually more than adequate, there are situations where higher performance is required. The byte code can be translated on the fly (at runtime) into machine code for the particular CPU the application is running on. For those used to the normal design of a compiler and dynamic loader, this is somewhat like putting the final machine code generator in the dynamic loader.

The byte code format was designed with this in mind, so the actual process of generating machine code is generally simple. Reasonably good code is produced: it does automatic register allocation and the compiler does some optimization when it produces the bytecode.

In interpreted code we're getting about 300,000 method calls per second on an SS10. The performance of bytecodes converted to machine code is almost indistinguishable from native C or C++.

> *When Archimedes was starting up, they did a prototype in SmallTalk. This impressed the investors enough that they got funded, but it didn't really help them produce their product: in order to make their simulations fast enough and the system small enough, it had to be rewritten in C.*

Multithreaded

There are many things going on at the same time in the world around us. Multithreading is a way of building applications that are built out of multiple threads[†]. Unfortunately, writing programs that deal with many things happening at once can be much more difficult that writing in the conventional single-threaded C and C++ style.

JAVA has a sophisticated set of synchronization primitives that are based on the widely used monitor and condition variable paradigm that was introduced by C.A.R.Hoare[‡]. By integrating these concepts into the language they become

† Threads are sometimes also called lightweight processes or execution contexts.

‡ 1974. Hoare, C.A.R. *Monitors: An Operating System Structuring Concept*, Comm. ACM **17**, 10:549-557 (October)

much more easy to use and robust. Much of the style of this integration came from Xerox's Cedar / Mesa system.

Other benefits are better interactive responsiveness and realtime behaviour. This is limited, however, by the underlying platform: stand-alone JAVA runtime environemnts have good realtime behaviour. Running on top of other systems like Unix, Windows, the Mac or NT limits the realtime responsivness to that of the underlying system.

- *Lots of things were going on at once in their simulations. Ropes were being pulled, wheels were turning, levers were rocking, and input from the user was being tracked. Because they had to write this all in a single threaded form, all the things that happen at the same time, even though they had nothing to do with each other, had to be manually intermixed. Using an "event loop" made things a little cleaner, but it was still a mess. The system became fragile and hard to understand.*
- *They were pulling in data from all over the net. But originally they were doing it one chunk at a time. This serialized network communication was very slow. When they converted to a multithreaded style, it was trivial to overlap all of their network communication.*

Dynamic

In a number of ways, JAVA is a more dynamic language than C or C++. It was designed to adapt to an evolving environment.

For example, one of the big problems with using C++ in a production environment is a side-effect of the way that it is always implemented. If company A produces a class library (a library of plug&play components) and company B buys it and uses it in their product, then if A changes it's library and distributes a new release then B will almost certainly have to recompile and redistribute their software. In an environment where the end user gets A and B's software independently (say A is an OS vendor and B is an application vendor) then if A distributes an upgrade to its libraries then all of the users software from B will break. It is possible to avoid this problem in C++, but it is extraordinarily difficult and it effectively means not using any of the language's OO features directly.

- *Archimedes built their product using the object oriented graphics library from 3DPC Inc. 3DPC released a new version of the graphics library which several computer manufacturers bundled with their new machines. Customers of Archimedes that bought these new machines discovered to their dismay that their old software no longer worked. [In real life, this only happens on Unix systems. In the PC world, 3DPC would never have released such a library: their ability to change their product and use C++'s object oriented features is severely hindered]*

By making these interconnections between modules later, JAVA completely avoids these problems and makes the use of the OO paradigm much more straightforward. Libraries can freely add new methods and instance variables without any effect on their clients.

JAVA understands the concept of an *interface*. An interface is a concept borrowed from Objective C and is similar to a class. An interface is simply a specification of a set of methods that an object responds to. It does not include any instance variables or implementation. Interfaces can be multiply-inherited (unlike classes) and they can be used in a more flexible way than the usual rigid class inheritance structure.

Classes have a runtime representation: there is a class named *Class*, instances of which contain runtime class definitions. From an object you can find out what class it belongs to. If, in a C or C++ program, you have a pointer to an object but you don't know what type of object it is, there is no way to find out. In JAVA, finding out based on the runtime type information is straightforward. For example, type conversions (casts) are checked at runtime in JAVA, whereas in C and C++, the compiler just trusts that you're doing the right thing.

It is also possible to lookup the definition of a class given a string containing its name. This means that you can compute a data type name and have it trivially dynamically linked into the running system.

> *To expand their revenue stream, the folks at Archimedes wanted to architect their product so that new aftermarket plug-in modules could be added to extend the system. This was possible on the PC, but just barely. They had to hire a couple of new programmers because it was so complicated. This also added terrible problems when debugging.*

Glossary

This paper is filled with all sorts of words and TLAs that my be hard to decipher. Here are the definitions of a few:

API — Application Programmer Interface. The specification of how a programming writing an application accesses the facilities of some object. Interfaces can be specified in JAVA and C++ using classes. JAVA also has a special interface syntax that allows interfaces that are more flexible than classes.

FTP — The basic internet File Transfer Protocol. It enables the fetching and storing of files between hosts on the internet. It is based on TCP/IP.

HTML — HyperText Markup Language. This is a file format, based on SGML, for hypertext documents on the internet. It is very simple and allows for the imbedding of images, sounds, video streams, form fields and simple text formatting. References to other objects are imbedded using URLs.

HTTP — Hypertext Transfer Protocol. This is the internet protocol used to fetch hypertext objects from remote hosts. It is based on TCP/IP.

Internet An enormous network consisting of literally millions of hosts from many organizations and countries around the world. It is physically put together from many smaller networks and is held together by a common set of protocols.

IP Internet Protocol. The basic protocol of the internet. It enables the unreliable delivery of individual packets from one host to another. It makes no guarantees about whether or not the packet will be delivered, how long it will take, or if multiple packets will arrive in the order they were sent. Protocols built on top of this add the notions of connection and reliability.

JPEG Joint Photographic Experts Group. An image file compression standard established by this group. It achieves tremendous compression at the cost of introducing distortions into the image which are almost always imperceptible.

Mosaic A program that provides a simple GUI that enables easy access to the data stored on the internet. These may be simple files, or hypertext documents.

RPC Remote Procedure Call. Executing what looks like a normal procedure call (or method invocation) by sending network packets to some remote host.

SGML Standardized, Generalized Markup Language. An ISO/ANSI/ECMA standard that specifies a way to annotate text documents with information about types of sections of a document. For example “this is a paragraph” or “this is a title”.

TCP/IP Transmission Control Protocol based on IP. This is an internet protocol that provides for the reliable delivery of streams of data from one host to another.

TLA Three Letter Acronym.

URL Uniform Resource Locator. A standard for writing a textual reference to an arbitrary piece of data in the WWW. A URL looks like “*protocol*://*host*/*localinfo*” where *protocol* specifies a protocol to use to fetch the object (like HTTP or FTP), *host* specifies the internet name of the host on which to find it, and *localinfo* is a string (often a file name) passed to the protocol handler on the remote host.

WWW World Wide Web. The web of systems and the data in them that is the internet.