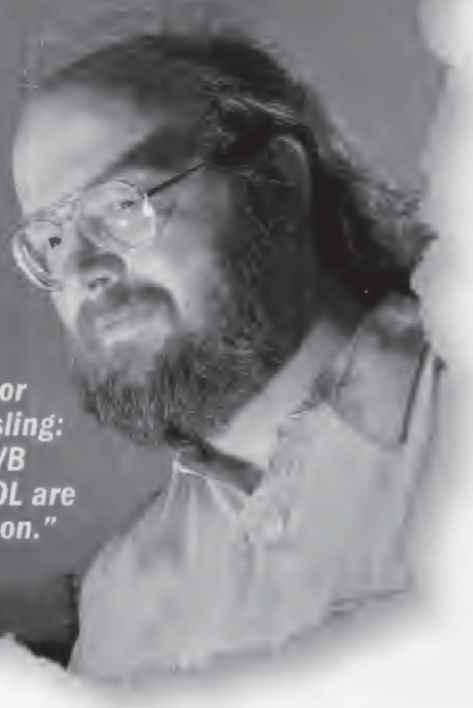
http://www.datamation.com

DATAMATION®

The Emerging Technologies Magazine for Today's IS

JAVA COMPLETE!



Java creator James Gosling: "Java for VB and COBOL are coming soon." p.30

?8
up

Exclusive: Richard Stallman Censors GNU Emacs, p. 98

```
#BXNDNOP ***************** 5-DIGIT 53711
#152469520/00638# (J6C01XX4) 002 0079
MICHAEL VAHLDIECK                198824
PROD. MGR.
EPIC SYSTEMS CORP
5301 TOKAY BOULEVARD
MADISON          WI   53711-1027
```

Also in this issue:

RPX Exhibit 1107
RPX v. AIT

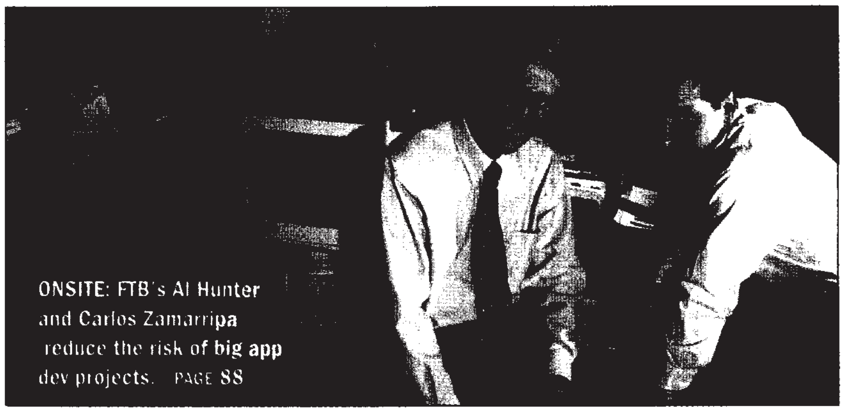
DATAMATION®

The Emerging Technologies Magazine for Today's IS

ADVANCED

COVER PHOTOGRAPHY BY ANDREW BRUSSO

March 1, 1996
Volume 42, Number 5



ONSITE: FTB's Al Hunter and Carlos Zamarripa reduce the risk of big app dev projects. PAGE 88

DATAMATION

http://www.datamation.com

J. William Semich, *Editor-in-Chief, 617-558-4261; Fax: 617-928-4261; bsemich@cahners.com*

Elizabeth Lindholm, *Executive Editor, 617-558-4308; Fax: 617-928-4308; elindholm@datamation.cahners.com*

Roger Plant, *Managing Editor, 617-558-4297; rplant@datamation.cahners.com*

Sarah Varney, *West Coast Bureau Manager, 408-345-4447; Fax: 408-345-4400*

Michael Alexander, *Special Sections Editor, 617-558-4612; Fax: 617-630-4612; malexander@datamation.cahners.com*

Ellen Greenblatt, *Administrator, Fax: 617-928-4281*

CUTTING EDGE

Susan Mael, *Associate Editor, 617-558-4438; Fax: 617-928-4438; smael@datamation.cahners.com*

DESKTOPS

David Simpson, *Senior Editor, 714-497-7377; Fax: 714-497-8749; dsimpson@datamation.cahners.com*

I-NETS

Sarah Varney, *West Coast Bureau Manager, 408-345-4447; Fax: 408-345-4400*

NETWORKS

Vance McCarthy, *Senior Editor, 408-345-4433; Fax: 408-345-4400; vmccarthy@datamation.cahners.com*

SERVERS

David Simpson, *Senior Editor, 714-497-7377; Fax: 714-497-8749; dsimpson@datamation.cahners.com*

SERVICES

Nancy Meachim, *Senior Editor/Research Director, 617-558-4453; Fax: 617-928-4453; nmeachim@datamation.cahners.com*

SOFTWARE

Mary Jo Foley, *Senior Editor, 206-523-5465; Fax: 206-523-6158; mfoley@datamation.cahners.com*

EVALUATION

Charles B. Darling, *Senior Writer, Voice/Fax: 813-443-7625; 701-9700@mcimail.com*

Susan Mael, *Associate Editor, 617-558-4438; Fax: 617-928-4438; smael@datamation.cahners.com*

CONTRIBUTING EDITORS

Elaine Appleton, *71271.2200@compuserv.com*

George J. Febish, *georgef@objectsoftcorp.com*

W. D. Riley, *wdriley@coh.org*

David E.Y. Sarna, *davids@objectsoftcorp.com*

Simon Travaglia, *spt@waikato.ac.nz*

COPY DEPARTMENT

Andrea Ovans, *Assistant Managing Editor, 617-558-4560; Fax: 617-928-4560; aovans@datamation.cahners.com*

ART DEPARTMENT

Sue Pulaski, *Senior Art Director, 617-558-4682; Fax: 617-630-4506*

Norm Graf, *Director of Creative Services*

PRODUCTION

Marty Moore

Assistant Managing Editor, Production, 617-558-4641; mmoore@datamation.cahners.com

Ana Murdock, *Production Editor, 617-558-4695*

Laurie Eleuteri, *Production Supervisor*

Heather Wiley, *Production Manager, 617-558-4522; Fax: 617-630-3930*

John Sanders, *Director of Production*

PUBLISHING

Carole Sacino, *Publisher*

Dan Savage, *Director of Custom Publishing*

Regina L. Twiss, *Marketing Manager*

DATAMATION *Address: 275 Washington Street, Newton, MA 02158, 617-558-4281. Circulation: 8773 South Ridgeline Blvd., Highlands Ranch, CO 80126-2329, 303-470-4445. Reprints/Permissions: Ellen Greenblatt, 275 Washington Street, Newton, MA 02158; 617-558-4281, Fax: 617-928-4281.*

Cahners Publishing Co., *a division of Reed Elsevier Inc.*

Robert L. Krakoff, *Chairman and CEO*

Stephen A. Thompson, *Sr. Vice President and General Manager, Boston Division*

Why Java?

By J. William Semich, Editor-in-Chief

IT WAS DURING AN EARLY evening phone conversation several weeks ago that it hit me–Sun's Java development language is a lot more than just another new toy for the Internet.

I was talking to John Patrick, an IBM VP, a.k.a "Mister Internet" among IBMers.

"What's your take on Java?" I asked, expecting the obligatory "We're working to support it in our browser" kind of answer. I had just that week finished putting up a couple of Java-animated hotlinks to advertisers' Web pages at PlugIn DATAMATION (http://www.datamation.com), you see, so Java and Web browsers were connected in my mind.

His answer surprised me. "IBM is porting the Java Virtual Machine to all our platforms," he said.

"Platforms?" I said. "You mean all the platforms your Web browser runs on, right?"

"No, no," Patrick answered. "We've already ported the Java VM to MVS, and we're working on ports to OS/2 and AIX. We're also going to port it to Windows 3.1, which Microsoft has declined to do."

"Java running on an IBM mainframe under MVS?" I asked. "Why in the world would anybody want to do *that*?"

This DATAMATION special issue–JAVA COMPLETE!–is our attempt to answer that question for enterprise users. To do it fast, and to do it right, the DATAMATION editorial staff has teamed up with a handful of technically savvy Java programmers and consultants who are about to launch a new technical newsletter, "The Java Report" (available, as they say, at a newsstand near you soon. Or you could **circle 150** on the Reader Service Card).

"Java running on an IBM mainframe under MVS?" I asked.

The result is the most complete enterprise-level view of this fast-paced new phenomenon available anywhere.

We start with a high-level overview in "Java: Internet Toy or Enterprise Tool?" (p. 28), followed by hands-on explanations of how to use Java to build cross-platform multi-tier client/server applications ("Business Uses for Java," p. 40, and "How Java Makes Network-Centric Computing Real," p. 42).

Then we dig deeper with detailed information on the Java programming language itself (p. 45), on Java's new memory management and security technologies (p. 47), and on Java's cross-platform-enabling Virtual Machine (p. 36). Not to forget the exclusive DATAMATION interview with James Gosling (p. 30), the inventor of Java.

And, for U.S. readers, we finish down in the engine room with a special 12-page Java tutorial, "The One Hour Java Applet," (p. 51).

So, if you've a mind to, you can get your hands dirty building a Java applet yourself.

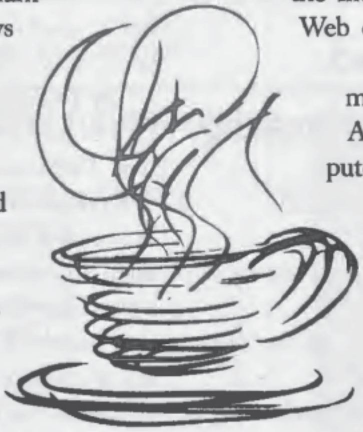
JAVA: Internet Toy or Enterprise Tool?

Java's a whole lot more than just an animation tool for the Web. Sun's renaming it the "Java Enterprise Platform." Better said, it's the Java network-centric computing platform.

BY BILL SEMICH AND DAVID FISCO

In January of 1984, during the Super Bowl's televised half-time ads, Apple Computer created in a single blow a whole new computing metaphor: the desktop computer as the key to personal creativity.

In Apple's ad, a young Olympic runner appears wielding an ominous hammer; she jogs past rows and rows of what can only be described as terminally brain-dead "suits" staring into green screens. She veers toward a billboard-sized image of Big Brother, and throws the hammer at the screen, shattering it and completely destroying computing's status quo.



Four days later, Apple Computer introduced the Macintosh and launched the era of desktop computing–in the enterprise as well as in the home.

In May of 1995, Sun Microsystems threw that hammer again when it loosed Java on the Internet.

Java–the name doesn't stand for anything–is actually two things: First, it's Sun's new simplified object-based, open-system language that allows software developers to engineer applications that can be distributed over the Internet using the World Wide Web or any of several other front ends currently under development by ISVs.

And second, it's a virtual computer (technically called the Java Virtual Machine) that will eventually allow all of the Java-based applications to become ubiquitous–to run anywhere and everywhere, regardless of the underlying hardware or operating system.

Thanks to the Java Virtual Machine, the same Java apps literally can run–unchanged and without recompiling–on anything from a smart cell phone to a laptop to a Windows 3.1, Win95, NT, OS/2, or UNIX workstation or server to an AS/400 or an IBM S/390 running MVS. It can do all this across any number of network and Internet protocols, using You-Name-It database gateways and with Whichever distributed object standards, including CORBA and Network OLE.

Revolutionary, no doubt. But even more revolutionary is Sun's decision to open up Java–and particularly to open up the Java Virtual Machine–to any and all. That means that *any* software vendors or individual developers–be they development tool vendors, language compiler developers, RDBMS vendors, middleware vendors, client/server application vendors or down-in-the-basement hackers–can use the Java Virtual Machine's bytecode language to create Internet-capable, run-anywhere applications and services.

So, just as the revolutionary new Mac hardware and OS combined to usher in the end-user-friendly era of desktop-centric client/server computing in the enterprise, all this new Java technology is now opening the way into the emerging developer-friendly era of network-centric C/S computing both inside the enterprise and out in the market.

The Java Revolution will deliver a world of software marked by individual empowerment. Java holds out the promise of making the operating system as transparent as a window pane. Software, information technology, and networked content will be judged by its quality and service, not by the proprietary (some might say "monopolistic") hold of its creator.

In a recent interview with Kim Polese, Sun's senior project manager for Java at the time, we asked, "What is Sun's business model for Java?"

She answered with one word: "Ubiquity." (Polese is now with a company called Yajsu, which was started by former members of Sun's Java development team, some of whom authored the material in "Java Complete!")

The Java community is becoming broader every day, encompassing some of the world's biggest independent software vendors, as well as users ranging from corporate CIOs, information technologists, system analysts, C/S developers, programmers, multimedia designers, marketing professionals, educators, managers, film and video producers, and even hobbyists.

Traditionally, not many of these people have worked together effectively. When we talk about open systems, we usually talk in terms of published APIs and specifications; in terms of availability of source code; or in terms of hardware, networking, and operating systems. But no one talks in terms of people being open. Java completes the cycle of an open system by getting all of these human resources on the same page. It opens up people and, in turn, opens up the way corporations work.

And it's for this reason that Sun's Java is gaining a place in the world of enterprise computing with incredible speed.

With this Special Issue, DATAMATION has partnered with the editorial team at the *Java Report*–a new monthly publication from SIGS Publications in New York City–to supply corporate IT decision makers with a complete guide to Java in the enterprise.

We start with an in-depth interview with Sun's James Gosling–the man who invented Java back in 1990–then take a business user's look at the technology, complete with a step-by-step explanation of how you can use Java to implement network-centric client/server applications for order/entry, accounts payable, and inventory management. Then we look at some technical details of Java that make it special–its virtual machine, its internal security and verification routines, and its simplified C++-like programming language. To top it all off, we've included a special 12-page DATAMATION Java Tutorial, so you can try your hand at building your first Java business applet in an hour.

When you're done, you should be able to better separate the Java hype from the Java reality.

J. WILLIAM SEMICH IS EDITOR-IN-CHIEF OF DATAMATION. DAVID FISCO IS EDITOR OF THE *Java Report*. FOR INFORMATION ON THE JAVA REPORT, CIRCLE 155 ON THE READER SERVICE CARD.

Run Your Enterprise on an OS for toasters?

THE ORIGINAL JAVA team worked on designing software for consumer electronics. They quickly found that languages such as C and C++ were not adequate because they have to be compiled for a particular computer chip. When a new chip comes out, most software has to be recompiled to make full use of new features on the chips. Once compiled, C and C++ programs are not easily adapted to use new software libraries. The programs have to be recompiled from scratch when the library changes.

Consumer device software has to work on new chips, though, because manufacturers are constrained by the cost of components. If the price of a computer chip becomes too high, they will replace it immediately with a newer, more cost-effective one. Even small price variations can make a big difference when selling millions of devices.

Software used in consumer electronics must also be very reliable, much more than most computer software. If a consumer product fails, the manufacturer usually has to replace the whole machine.

Gosling On

Senior Editor Vance McCarthy caught up with Sun Fellow and Vice President James Gosling just hours after Sun released the final version of the Java lanuage, development kit, and runtime engine (the Java Virtual Machine) across the Internet. Gosling, who wrote the original source code for UNIX emacs, first came up with the design for Java back in 1990. Today he heads up the technology arm of Sun's whole Java development operation.

DATAMATION: Can you define or explain Sun's plans for or expectations of Java?

GOSLING: Let's see. There's a of string of [Java] buzzwords out there about building sort of reliable, multithreaded applications that can rove around the network in a safe way.

There are a lot of mechanisms in Java to let you sort of transparently move behavior around. And it's not something that users will see directly, and they won't be afraid that there's some virus coming at them that's going to take over their system.

DATAMATION: You said "around the network," not the Internet. You make it seem as if Java is not expressly an Internet technology.

GOSLING: No. It's a general-purpose programming language. And people have used it for all kinds of stuff.

A lot of what's interesting around it was driven by what it takes to build applications on a network, but that's not all of it. There are lots of folks who are perfectly happy to have something that is a cleaned-up version of C++ with multithreading and garbage collection.

COMPLETE

JAVA

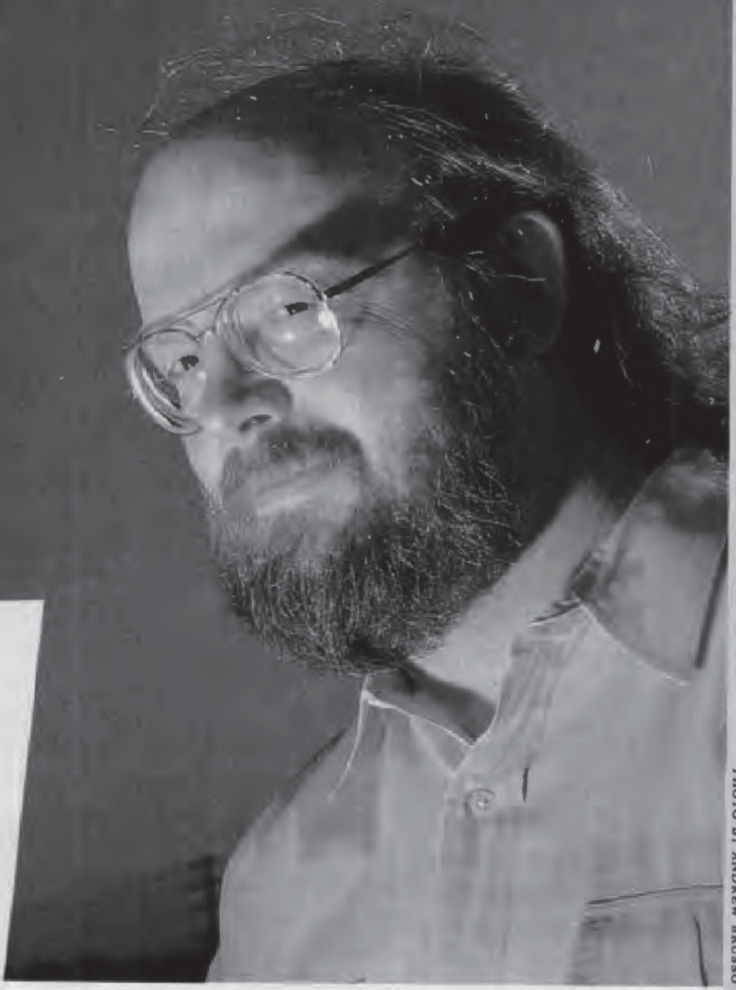


PHOTO BY ANDREW BRUSSO

Why Is It Called Java?

The Java language was originally called Oak by James Gosling. His inspiration for this name was a large oak tree outside his office window at Sun Microsystems.

Later, the Java development team discovered that Oak was the name of a programming language that predated Sun's language, so another name had to be chosen. It's surprisingly difficult to find a good name for a programming language, as the team discovered after many hours of brainstorming.

Finally, inspiration struck one day during a trip to the local coffee shop.

DATAMATION: Okay. Can you define the difference between a Java applet environment and a distributed client/server environment? How would Java operate for the end user?

GOSLING: What most people do is use this through something like a Web browser.

So what folks often do is they have a company home page that's got pointers to various [Java] applications. So, when somebody wants to issue a petty cash request, they go to the petty cash page and fill it in and interact with the stuff that's there.

And when there are new services out there on the network, you get to them through the normal Web navigation tools–you can now do expense reports on line [with Java]. You click the link, and there you are–the expense-reporting tool.

DATAMATION: In terms of that kind of seamless access for the user, what's going on behind the scenes, either in terms of a core Java technology or what a developer might have to do with Java to get that to work?

GOSLING: Well, the developer basically writes the client side and installs that on the server. And when the client says, "Gee, I want to look at the petty cash page," the page comes over. The page has [a Java applet] embedded in it that says "Gee, this rectangle on the page is supposed to be occupied by the petty cash application. Do I have a copy of it locally? Well, no, so I better go back to the server and get a copy of it." It comes over.

That's sort of extra–that sort of transaction, bringing it over–and then, once it comes over, first running it through something called the Verifier, which checks out a whole lot of stuff. Then it actually gets run. And all of these steps of bringing the application over, verifying it, and the rest of it, are completely transparent.

The magic that happens is that I'm sitting here in my site station in Palo Alto. I just clicked on a link to a Web page. This went to the University of Syracuse. The Web server at Syracuse sends this thing over, and it's embedded in the middle of a document. And I didn't have to get a copy of the software, copy it over, install it on my machine, worry about running some virus scanner on it. It just happened. So the network appears completely transparent.

I don't have to have authenticated these guys [the applets]. I don't have to know whether or not I can trust them, because the application is essentially run inside a box that sort of limits what it can get at. So that it's just a programming language inside. The stuff that you can write in C++ you can write in this. And the stuff that you could write in Java you can also write in C++.

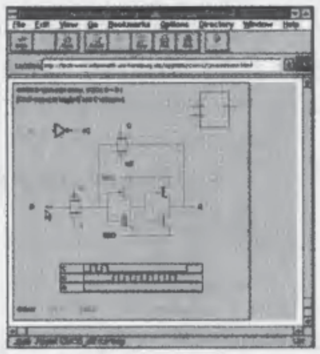
What it gets you is the ability to have these pieces of code sort of rove around the network. If you're somebody who has got some kind of data in some special form, and you want to publish it to the world, you just put it on your Web site and tell people to go look at it. And you're not limited to what you can do with a normal Web tool. A normal Web tool gives you flat pages that are text and images, and it's sort of like looking at a book.

DATAMATION: How does Java change that?

GOSLING: Java lets you put arbitrary dynamic stuff in there. I mean, a lot of people are using it as though it was sort of a fancy way to do [interactive] illustrations in books. There are lots of folks who do things like use Java for textbooks. What gets embedded in the pages is like a simulator for something–like a simulator for an oscilloscope rather than a photograph of a face on the oscilloscope, or a circuit simulator instead of a circuit diagram.

DATAMATION: Can you actually put the functioning circuit on a diagram in there?

GOSLING: Right. And you can actually interact with it. So you get this circuit diagram in the page, and you can click on the circuit diagram, and things happen. The best one of those was done by some



Java-interactive circuit design at http://tech-www.informatik.uni-hamburg.de/applets/cmos/cmosdemo.html

Move over ODBC! Here Comes JDBC

Every third person on the net wants database access. They're about to get it—Java style. Next week, JavaSoft will release a spec for Java Database Connection (JDBC), an API that lets developers write Java apps that access databases. To support interchangeable DBMS drivers, the API uses a driver manager that automatically loads the right JDBC driver to talk to a given database. JavaSoft will bundle the JDBC driver manager into future releases of Java. The company is also going to release a software bridge from JDBC to Microsoft's ODBC database driver interface.

According to JavaSoft president Alan Baratz, the driver is the first of several products in what JavaSoft calls "the Java enterprise platform," —a platform JavaSoft hopes will turn Java and the Web into an enterprise-class platform for applications and information access.

The initial spec will be released on March 8 at http://java.sun.com. Look for the final spec and software this summer.—*Susan Mael* guys in Germany. And I just had to click on their Web page, the Web page comes over, and here's this thing with circuit simulation in it.

Somebody has to write that. And what tends to happen a lot is many applets get used over and over again. So, there's one applet that draws a chart, and the chart picks as a parameter the URL to some data source or data stream, if it's going to be a live chart. And it puts the data in the chart [while its changing]. One of the nice things about doing the chart application as an applet is that it stays live. So if it's connecting to a database server that's got stock prices, then you get this chart up on your screen, and it's got the stock price diagram. But as the stock price changes, you see the graph move. [See related article in this issue, "How Java Makes Network-Centric Computing Real," page 42.]

DATAMATION: It seems as though Java is very operating-system independent or is capable of being that way.

GOSLING: True. You don't [need to] know whether somebody is running on a Mac, a PC, or any old flavor of UNIX.

DATAMATION: Right. But what about the server side—is Java database-server and network-OS-independent, too?

GOSLING: It depends. With things like databases, like Oracle and whatever, there's a standardized API [see sidebar, "Move over ODBC, Here Comes JDBC," page 32] that is,

The Java-based "Internet PC"

EW INDUSTRY TOPICS HAVE caused as much debate as the Internet PC. The Internet PC is championed primarily by IBM, Oracle, and Sun, but a variety of smaller vendors is also getting into the act.

The central idea is that these Internet PCs will be under-$500 devices that people can use to download programs from the Internet whenever they need them, without storing programs and files locally (most Internet PC designs do not include a hard disk). For IS managers, the Internet PC promises significantly reduced ad-ministration, maintenance costs, and time; the elimination of costly upgrade cycles; and loads of free or almost-free software.

In the Sun vision of the Internet PC, Java runs as a front end to a tiny operating system—possibly called Java OS—that is similar to a microkernel. The operating system code will be able to run on a variety of microprocessors and on devices ranging from handheld PDAs to low-end PCs with minimum memory—probably 4MB.

If Java succeeds in enabling programmers to write very compact code, then it will be possible for users to download and run Java applets reasonably quickly, despite the constrained bandwidth most users face today.

In January, Sun previewed at the Demo '96 trade show a prototype of an Internet PC that measured about 5x9x2 inches, excluding the monitor. Sun officials admit that the machine will cost more than $500, mainly because it includes a workstation monitor.

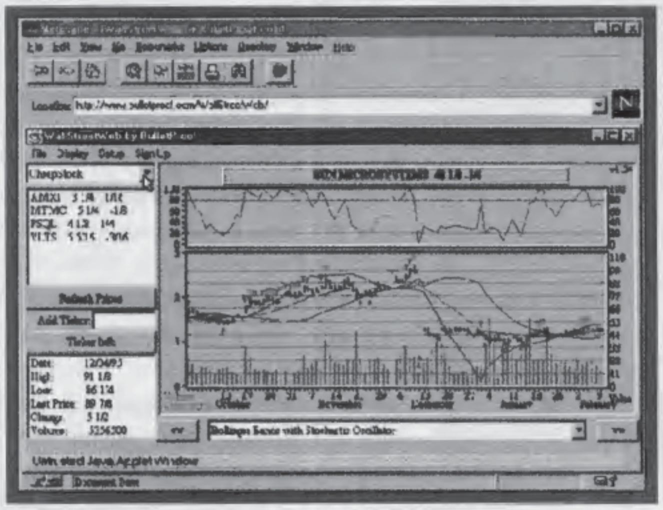
For an idea of what the under-$500 devices will look like, consider Oracle's Network Computer (NC). The base model will include a low-cost RISC microprocessor, a ROM-based operating system, 4 to 8MB of RAM, a PC Card expansion slot, an Ethernet connection, a keyboard, and a mouse. Because the proposed system doesn't include a monitor, users would have to plug it into either a PC monitor or a TV.

Prototype of Sun Microsystem's Internet PC.

A prototype of Oracle's NC is due this month, and limited shipments are expected in late summer. Oracle plans to design a portable unit and a more expensive multimedia-capable unit. The company will license the designs to other manufacturers.

Although Sun, Oracle, and IBM (with its InterPersonal Computer, or IPC) get most of the attention, a handful of smaller vendors are expected to ship Net devices this month. Examples: Acorn Computer Group in Cambridge, England; SunRiver Data Systems of Austin, Texas; and TransPhone of Ottawa. Late this year, The WebBook Co. of Birmingham, Mich., plans to deliver a Web-access device designed specifically for Java.

The device will use the ShBoom RISC chip, from San Diego-based Patriot Scientific, as its Java runtime engine. Like Java, ShBoom uses a stack-oriented architecture, which Patriot officials claim will give it a speed advantage over other processors running Java. —*David Simpson*



· **Java-based real-time stock-market-monitoring system at http://www.bulletproof.com/WallStreetWeb/**

not too surprisingly, relatively similar to ODBC. And so we build these sorts of standardized APIs that try to give you the same view of everything everywhere. So an Oracle database looks the same no matter where you are. What we try to do is provide as large a suite as we can of APIs that are portable across all platforms.

DATAMATION: Where are Java developers focusing their development activities right now?

GOSLING: Well, for the most part, people have been doing client-side stuff–where they talk to some existing server-side application, and they're really using Java to build a Web-based front end to their application.

DATAMATION: Would you say that Java's architecture will incorporate some core Operating System services in its environment?

GOSLING: Well, there are a number of core services that we do ourselves, although they tend to be relatively limited. In general, the policy is to try as much as possible to leverage off of whatever services are on the platform.

I think people are only beginning to understand what networks really mean. For example, people have tended to build network security systems that are still somewhat tied into the time-sharing system metaphor, where you have a server and log onto the server and you do stuff, rather than treating the network as a more homogeneous sea of computation. But, as people start to figure out what that really means, I think a lot of these services will change.

DATAMATION: Let's take an example, say a stock-market-reporting service or maybe some other more elaborate sort of real-time transaction-processing environment. Are there things that Java has to do to make sure that those real-time access services might be equally available to any user on any platform?

GOSLING: Yes. Well, what ends up happening with some of the very specialized uses like that is that we tend to build an API to something like Reliable Transaction Service. And we try to get people to standardize on it. If you're doing a reliable transaction, you try to expose at least the common features through this standard API. And that seems to work pretty well, and we've got lots of mechanisms for allowing people to add stuff to it. But, for some of these, even though there's a standard API, that doesn't mean that the API must necessarily be available on all machines.

Because one of the things that a Java application can do is to inquire of its environment, "Do you support this interface?" and get the answer back, yes or no. So if you've got an applet that needs to use the secure transaction system, it can ask, "Is the secure transaction system here or not?" And if it's not, then you can back off and try another strategy.

DATAMATION: Could you

Java's Guarantees

Java is a language with a rich set of guarantees that, when taken together, provide a truly advanced operating system for programs. Most of these guarantees are also available in other languages, but no other language or operating system provides all of them together.

1. Java code is portable and will run without alteration on almost all modern operating systems.
2. Java provides true multithreading to the programmer, along with synchronization methods.
3. Java provides platform-independent user interface classes.
4. Java comes with built-in networking.
5. All Java objects know how to print themselves out.
6. The Java compiler enforces all error and exception handling.
7. Memory is cleaned up automatically.
8. Programs written as applets can automatically be distributed across a network.
9. Java applets will run seamlessly in a lightweight client Web browser on any platform.

Source: Hooked on Java by Arthur van Hoff, Sami Shaio, and Orca Starbuck (Addison-Wesley Publishing).

actually put an automatic redirector on the server to say "Well, then, you want to go here?"

GOSLING: Yes, and a number of people have built things like that, where, like on the client, it will ask, "Gee, can I do thus and so?" And if the answer is no then it says, "Okay, I'll try a different strategy for this." Which may be falling back to a different way of accomplishing the same thing locally or like talking to another server. So if you don't have an Oracle interface to such and such a database–and it's one of these distributed Oracle databases and you wanted to make a query–then if you know some other host on the network that does have the interface and is willing to talk to you, you can establish a connection and say, "Please do this query on my behalf." And then that guy [the applet] turns around and makes the query to Oracle and sends it back. So you essentially sort of redirect it.

"I think people are only beginning to understand what networks really mean."

DATAMATION: What's the best way for users to introduce Java into their enterprise C/S systems?

GOSLING: What most people are doing right now is they've got some client/server application, like a travel expense system or whatever. And they keep that running, and then they write the Java/Web style of the front end, and they put that out onto the Web server and let people use it. As that matures you run the two client-side systems in parallel. And at some point you make a decision and turn one off.

DATAMATION: So I really don't have to worry too much about fiddling with my back-end data repository to make it Java friendly? Because of your APIs?

GOSLING: Yes, the way that the APIs work is that a lot depends on what the back end is. There are adapters to a variety of different kinds of back ends. Like there are adapters for, pick your favorite database engine, Oracle, whatever.

DATAMATION: All the SQL engines?

GOSLING: I think that if you poked around, you could find essentially all of them. I know that there are a couple of companies that specialize in providing these sort of data access adapters. Web Logic is one of my favorites. Plus there are a set of really low-level APIs that–I mean, one of the things about a lot of these legacy databases is that they speak just astonishingly weird protocols.

DATAMATION: Microsoft seems to be trying to position Java as just kind of a VB-type environment that

Is Java Better than OLE?

SUN IS NOT THE ONLY company to think about distributed objects. Microsoft has had a great deal of success in promoting its Object Linking and Embedding (OLE) specification as the de facto standard for software components. Furthermore, the network-enabled OCX (i.e., OLE control) that will be shipping with the next version of Windows NT (NT 4.0) resembles Java objects in many respects. Both support an important object-oriented feature known as inheritance. This ability to inherit functionality from a parent object is critical if network objects are to fulfill their potential.

But even though Java applets and network-enabled OCXs are similar in many respects, there are some very important distinctions.

First of all, Java applets are platform independent. A Java applet can be written once and run on any PC running the Java Virtual Machine. While a Java applet is platform independent, an OCX is more or less limited to Windows-based workstations.

Second, Java applets execute within the secure Java environment. In contrast, users that execute a spreadsheet OCX originating from one of the potentially millions of Internet nodes have no assurance that what they're actually running is not a virus bent on wiping out their hard drives. Java is different. It does not allow applets to have unauthorized access to memory or hardware, thus greatly limiting the potential damage that a malignant program could cause. (See related article, "Yes, Java's Secure. Here's Why," page 47 .)

Finally, an important difference between an OCX and a Java applet is that an OCX only goes in one direction: from the server to the client. Java applets can go in both directions, blurring the distinction between client and server. For example, a client wishing to search through a database spread across multiple servers could dynamically send an applet to each server that will do the work. Again, these applets are different from traditional client/server applications in that they do not have to be preinstalled.

—Jim Flynn & Bill Clarke

JAVA SUPPORTS MORE NETWORK-CENTRIC FEATURES THAN THE OCX

	Java	OCX
Can run over a network	Yes	Yes
Supports inheritance	Yes	Yes
Platform independent	Yes	No
Runs in a protected environment	Yes	No
Bidirectional between client and server	Yes	No

Source: @Work Technologies

emerged out of the UNIX world–no big deal. Do you think Microsoft might just support Java in a limited way in some environments but not fully exploit it across the board?

GOSLING: They're [Microsoft] being rather disingenuous about a number of things. They've always had a very casual attitude towards security and viruses. All of their systems are just designed to host viruses. I mean, it's like a petri dish with the best culture you could buy. And all the issues about portability, of course, don't matter in the Microsoft world because there's only one platform. I mean the license–there's the [letter of intent between Microsoft and Sun] that they signed. It was just about running applets inside the Internet Explorer. Basically just sort of extending their Web browser.

DATAMATION: Are there more aggressive things that Microsoft could do in Windows95 or in its NT suite that would extend Java capabilities beyond what the Java APIs and your OEMs could enable?

GOSLING: Well, they could do things like put [Java] in the Windows95 kernel, which would make [Win95] accessible by all [Java] applications.

DATAMATION: And what about the server side? Putting Java into the NT Server or Microsoft's Web server–what might that do?

GOSLING: You can do all kinds of stuff by putting Java in the Web server–or in an ordinary database server. Lots of folks have been doing things that are very similar to the sort of agent stuff that General Magic used to talk about, where you basically download a piece of intelligence into a remote server and say "go do this for me."

DATAMATION: Tell us about Sun's vision of the Java Virtual Machine and how you think Java will exploit that concept.

GOSLING: Well, the Virtual Machine is sort of the specification of this "fictitious CPU." And this fictitious CPU has a number of interesting properties. One is it's real easy to write an interpreter for it. Another is that it is very analyzable–you can analyze the instruction set and make certain inferences about how it's going to behave. That's one of the foundations of the Java security system. Another that's related to being analyzable is the issue of being able to take that Virtual Machine instruction set and translate that sort of cheaply and easily on the fly into machine code. So when the Java bytecoded thing [an applet] shows up on the machine, and it's been shipped from Zimbabwe up to my machine, and I'm running on a Solaris SPARC chip, it can get translated on the fly into machine code and run at essentially full C-code speeds.

The Java Virtual Machine: A Soft-CPU

PROBABLY THE MOST important element of the Java environment is what James Gosling decided to call the Java "Virtual Machine." The VM is a whole lot more than your typical BASIC or other high-level language runtime interpreter. The VM is modeled on a small, efficient CPU—it takes Java-compiled bytecode and runs it as if it were machine language. The lower level of the VM converts this pseudo-machine code to actual hardware calls (working with whatever OS is resident on the machine it's running on).

The Java VM supports 248 or so of these bytecodes. Each performs a basic CPU operation like adding an integer to a register, combining the numbers in two registers, jumping to subroutines, storing a result, incrementing or decrementing registers—you name it. The VM is, in effect, a stacked arithmetic logic unit (ALU) with local and global variables.

Here's how the stack architecture works: To add two numbers, the VM first pushes them onto its stack, then adds them. After completing the addition, the VM leaves the result on the stack for the next step in the process. If you think of the logic of an HP calculator, you'd be close to understanding the VM's logic.

Local VM variables are used for temporary storage and calls to subroutines. Global variables are used to keep track of, for example, where the actual machine code from the Java bytecode program is executing in memory, what the current object is, where the relevant data is stored in memory, and other kinds of global information.

It should come as no surprise that some chip makers (including Sun Microelectronics and Patriot Scientific) have implemented the Java bytecode as the basic instruction set for a new breed of CPU. These can be used for simple handheld devices (cell phones, pagers, etc.), as well as Internet-specific devices.

By using bytecode operands burned into the silicon itself, Java can run at blazingly fast speeds.

—J. William Semich

One of the things that we're trying to–changing direction completely here–is that the Virtual Machine is pretty much tuned to just doing Java, which is a sort of variant of C++.

DATAMATION: Does that mean Java is just a subset of C++?
GOSLING: Not so. We're looking into what it takes to support other languages on top of the same Java Virtual Machine. And amongst those are things like Visual Basic, some of the list languages like Dylan and Scheme. Some folks have already done an Ada compiler that targets the Virtual Machine directly. With the Java Virtual Machine, it isn't so much the Java language that's involved in the Net but it's really the Virtual Machine layer and any language. That's where all the transport around the network and the adaptation to arbitrary machines happen.

So, yes, you can actually put other languages besides Java onto the same Virtual Machine.

DATAMATION: But just in theory at this point, right?
GOSLING: In theory, yes, but a few have actually been done. Like there's this company called Intermetrix, which does Ada compilers and stuff. And they've announced an Ada compiler that generates bytecode for the Java Virtual Machine.

DATAMATION: And the purpose of somehow introducing a Virtual Machine architecture into your distributed system would be to do what?
GOSLING: That's really the foundation of all this stuff about code migrating around the Net and all the security mechanisms [you need to be able to do that]. And if you can layer different languages on top of it, different people are comfortable with different languages for a variety of reasons.

The Java language comes from a C++ tradition. But one can easily imagine some people being more effective using languages from a list tradition.

DATAMATION: Would you see this being of interest to folks that are in the COBOL tradition?

"With the Java Virtual Machine, it isn't so much the Java language that's involved in the Net but it's really the Virtual Machine layer and *any* language."

GOSLING: Oh sure. Compiling COBOL into the Java Virtual Machine I think is probably not hard. I've got a fairly reasonable idea of what it takes to get COBOL into a machine, and I think COBOL is a fairly reasonable bet for the Java VM.

DATAMATION: How does the Java Virtual Machine work, exactly?
GOSLING: When you go to run any Java program, whether there's a compiler, or somebody's development environment, or an applet you've developed or whatever, you run this [Java] Virtual Machine interpreter.

DATAMATION: Some of the folks in the computer industry and the press are saying, "Well, Microsoft has finally met its match with Java." What would you say to those that see the Java Virtual Machine as being perhaps a "Microsoft Killer" technology?
GOSLING: Well, I mean, I think that's a fairly daunting proposition. Sometimes, when I see some of the articles that people write, I sort of feel like I'm set up for failure. But I think the real point is that–

DATAMATION: Into the jaws of Redmond, do you mean?
GOSLING: Yeah, like somebody's been coming out with a pot of red paint and painting circles on the top of my head. But I think the real truth is that they're very different things [Windows vs. Java]. That they have different goals.

DATAMATION: You mean, basically, that the core reason for the Virtual Machine is purely technical–to guarantee platform independence for Java?
GOSLING: Right. It's just sort of the mechanism, and it's really centered around building these pieces of behavior that migrate around the Web. And, yes, you can load it up with all of the stuff to make it a full-fledged operating system, and there are some people who are bravely off there doing it. But that doesn't mean the goal was ever to supplant Windows or supplant UNIX or supplant the Mac OS. The goal was rather to provide sort of a unifying view on top of all of them.

Business Uses For JAVA

THINK CLIENT/SERVER COMPUTING AND THE INTERNET HAVE CHANGED YOUR LIFE? JAVA'S GOING TO BE TWICE AS IMPORTANT—IT WILL TOTALLY CHANGE THE WAY YOU WRITE APPLICATIONS AND DISTRIBUTE THEM TO CLIENTS.

BY JOEL SCOTKIN

UNIX MAY NOT have taken over the desktop, but some of its most important core ideas have found their way into all of the other major operating systems and technologies in the enterprise. UNIX networking, in particular, and the distributed-computing environment it has engendered, are now staples of the corporate-computing environment.

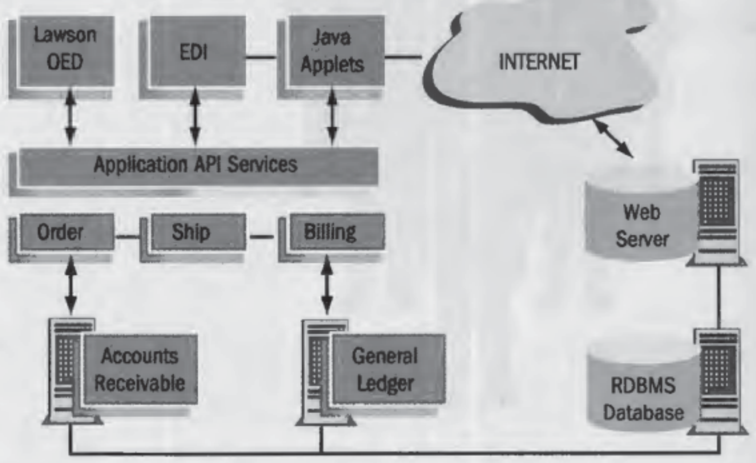
Any successful new technology eventually leads to even newer ways of doing things, and the distributed-computing model is no exception. The first child of this model, client/server computing, has revolutionized the way that businesses develop and use applications. The second child, the Internet (perhaps more important, its internetworking protocols and standards), has revolutionized the distribution of information.

Now comes Java, along with several concepts surrounding it, the third child of this computing model. But Java could have as great an impact on the distributed-computing model as the first two offspring combined.

Java reinvents the way applications are distributed to clients and executed, thereby creating a world where business applications can be shared with clients without security risks, where small snippets of code can be sent along with raw data to create dedicated information filters, and where universally portable code allows for new kinds of computing devices.

A Java-Based Multitier C/S System

Lawson Software's OED uses Java applets to do transactions across the net



Source: Lawson Software Co.

Java stands out in three areas of enterprise distributed computing:

- **CLIENT/SERVER APPLICATION DEVELOPMENT:** Java is not just an Internet browser add-on. It is a new, simpler way to develop internal client/server applications. Java is an excellent alternative to using C++ to develop distributed applications that run over the corporate LAN.
- **DATA ACCESS:** Java is an easy way to deliver business information broadly, potentially both internally and to external customers simultaneously (data access over the Internet). This might involve using Java to write custom viewers for different data sets, thereby letting users get the most from your information (e.g., on-the-fly charting of raw data or interactive queries or filters on large data sets). Data can also be wrapped in spreadsheet applets so users could analyze the information on the fly.
- **ON-LINE TRANSACTIONS:** Java can be used to develop true transaction-based applications, either for business-to-business or business-to-consumer interactions. This is the

electronic commerce model, currently a relatively small market that could begin to grow exponentially.

All three of these uses share the client/server trait of separation of information from user interface. Java is an ideal language for writing the client side of these client/server applications. The built-in classes let the programmer build interfaces in a fraction of the time it would take using Microsoft Windows or X/Motif. The networking encapsulation takes care of remote connections; and the elegant objects leave the code easy to extend, as needed.

The inherent portability means you can distribute your application throughout your company, in many cases just by putting it up on an internal Web page. Just as easily, you can let selected clients (or the world) run your application. For an excellent example of a Java-based multitier client/server application, see related story, "How Java Makes Network-Centric Computing REAL," page 42.

The bottom line is that Java can help you get your information out faster. Use Java for internal projects and develop applications faster, then run them on all of your platforms at once. Using Java can drastically reduce the effort needed to install, support, and upgrade applications–the latest version will always be one mouse-click away. Think about instant distribution of a new application to your business offices around the world! Or use Java to write Internet applications and get a jump on your competition.

For instance, if you run a business where your clients call up a support person who then queries a database and gives them a verbal response, you could create a custom interface to your data and make it more easily accessible to the client. Much of my work is done with the financial community, and I have already seen market data companies creating live interfaces to stock and bond prices. I've seen banks creating Internet-based trading systems. And I've seen internal applications being rewritten in Java so that they can be shared with preferred clients. Any business that makes a living selling information or services can benefit by a better way of getting out information updates, and Java can provide that way.

Java also works right out of the box (or right off the Net). C++ was hobbled for a long time because of a serious dearth of development tools. Products like Purify and RogueWave helped make C++ a serious language for application development. The main downside to Java is that there are currently no development tools available, but the built-in language facilities have lowered the application development hurdle enough that Java can already compete as is and will be absolutely stunning when the tools are available.

I am recommending to my current clients that they use Java as part of a three-tier architecture (see the diagram). The idea is to separate new applications into three parts: the information source (back-end databases), the part of the application that does heavy calculation (the application and Java applet server), and the user-interface client (the Java Virtual Machine).

This separation of components makes sense no matter what the development language, but also presents an easy way to port pieces to Java–a clean way to distribute applications or to safely write Java prototypes on top of existing applications. It also offers better security for your data, leaving only a lightweight interface in the hands of your clients.

Java is easy to learn, speeding both prototyping and development time for a completed application. Spend a few minutes, and you'll almost certainly come up with an example of how it can benefit your business. Spend just a little more time, and you will be able to test whether or not you were right.

The Java Enterprise Platform

SOME ENTERPRISE USERS aren't waiting to use Java to set up distributed business systems across the Internet. Ingenius Co. of Englewood, Colo., for example, is in the process of using a Java-enabled version of Lawson Software's Open Enterprise Financials and Distribution Management systems to provide an à la carte order-entry service on its Web home page.

Ingenius offers an on-line news service oriented for children. The Lawson on-line system, which runs on a Digital Alpha server using Informix, lets customers fill out an order form, download their images, and receive an invoice in the mail.

Several other enterprise software vendors—including Adobe Systems, Borland, Gupta/Centura Technologies, IBM, Microsoft, NAT Systems, Oracle, Peoplesoft, SAP, SAS, Seer Technologies, and Symantec—are developing Java implementations of their software and development tools. IBM has even written a Java Virtual Machine for its mainframe MVS operating system. *—J. William Semich*

JOEL SCOTKIN IS PRESIDENT OF RANDOM WALK COMPUTING, A START-UP DEVOTED TO JAVA CONSULTING IN THE NEW YORK AREA. AS A CONSULTANT TO THE FINANCIAL COMMUNITY, HE BUILDS DERIVATIVES-TRADING SYSTEMS AND DESIGNS AND IMPLEMENTS INTERNET FIREWALLS AND WWW SERVERS. CONTACT HIM AT AT JOELS@RANDOMWALK.COM.

How JAVA Makes Network-Centric Computing REAL

JAVA IS LIKELY TO HAVE A PROFOUND IMPACT ON THE WAY BUSINESSES MANAGE APPLICATION DEVELOPMENT BY BRINGING ABOUT AN ENTIRELY NEW MODEL—THE NETWORK-CENTRIC MODEL—OF C/S COMPUTING.

BY JIM FLYNN AND BILL CLARKE

JAVA'S UNIQUE CAPABILITIES, working with other Internet-based technologies, make it possible to create distributed object-oriented applications that exist and function independently of any particular desktop architecture. Consequently, Java is well on its way to becoming the preferred development environment for an inherently more flexible and dynamic computing model.

Most common client/server applications today are founded on the desktop-centric model of computing. In other words, applications are designed with a specific client architecture in mind (e.g., Mac, OS/2 Warp, UNIX, or Windows). These desktop-centric applications must be preinstalled, or effectively "hardwired," onto client PCs. To make any changes, you have to install all-new software or upgrade on a client-by-client basis.

The World Wide Web and the Internet provide the basis for a technology that's changing all that. Platform-independent standards such as HTML and TCP/IP let Web developers build a single "application" (Web page) without worrying about operating systems or hardware platforms. At the same time, users need not be concerned with the type of server being accessed. This, more than anything else, has contributed to the Web's explosive popularity.

And now, with Java, Web servers can transmit much more than just static HTML data out to client users; they can also transmit a series of small programs called "applets." These applets can be strung together using various scripting languages to become full-fledged applications.

A Java C/S app looks and feels pretty much the same as any other C/S app.

For example, if a client requests spreadsheet data from a Web server, the information is transmitted with a Java wrapper applet that can run the spreadsheet.

This bundling of data and associated Java programs into objects is what makes network-centric computing possible.

A big advantage that network-centric Java applets have over desktop-centric applications is that they don't have to be preinstalled–they install themselves just in time, on the fly, and deinstall themselves when they're no longer needed. Hence, the concept of upgrading application software is eliminated.

A prerequisite for such a Java-based app is that your users have a Java-capable browser on their workstations. The workstations could be a Windows PC, Mac, OS/2 Warp, UNIX workstation or thin client (Internet device). Sun's HotJava browser and Netscape Communications' Netscape 2.0 are already Java enabled. And you can expect virtually all other Web browsers to soon follow.

A good example of the kind of network-centric business application you could build with Java applets is an order-entry system. The order-entry process is fairly simple; however, several relatively small steps are involved. Each of these could be a Java applet.

In a Java environment, all of the order-entry application software would be transmitted dynamically to the client workstation as needed. The application is not only broken down by client-based applet but may even be spread across multiple servers.

As far as the end user is concerned, a Java-based client/server application looks and feels pretty much the same as any other C/S app. Upon connecting to the order-entry server, the user is presented with a security log-in. The encryption of the log-in and password may be handled by the browser/server environment or by local network protocols.

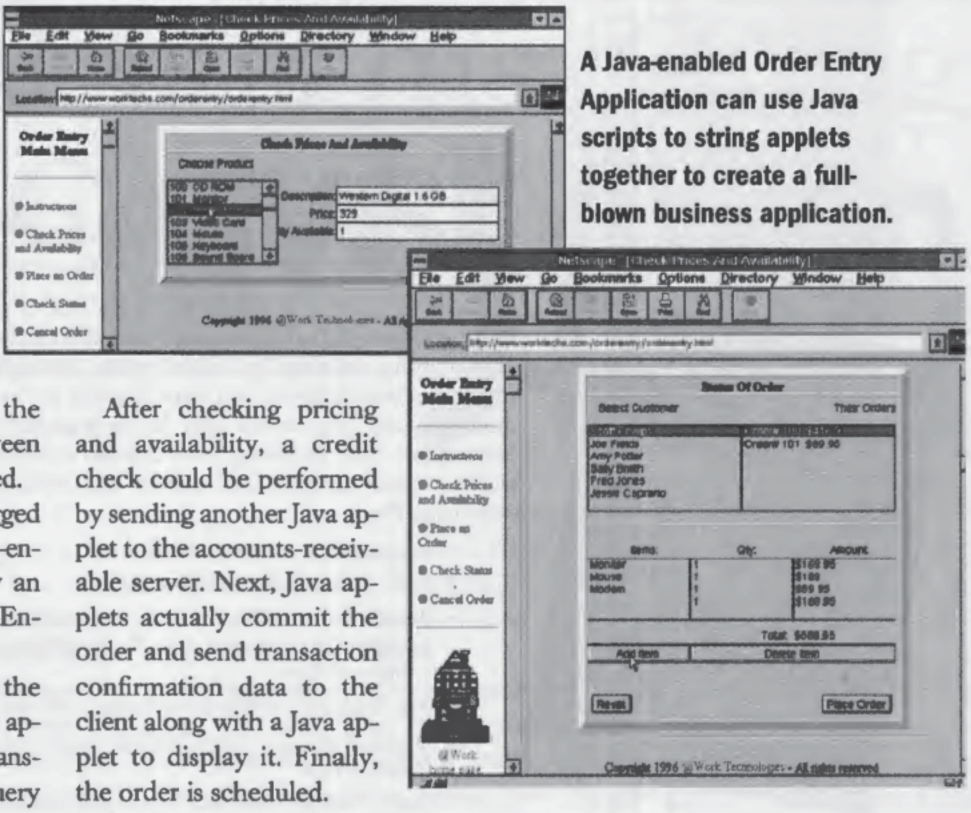
Java also provides predefined code, called classes, that can handle security (see related article in this issue, "Yes, Java's Secure. Here's Why"). In addition, the ability to use public and private encryption keys will soon be released with version 1.1 of Java. The use of encryption would help reduce the likelihood that the transmission of sensitive data between network nodes could be intercepted.

After the user has successfully logged in, the Java applet displays an order-entry main menu, which is actually an HTML page (see figure, "A Java-Enabled Order Entry Application").

Each nested HTML page on the opening page contains appropriate applets to complete the order-entry transaction, verify input, and display query results from a server-based database system, which can be invoked via server-based applets.

One of the key advantages of using Java for network-centric computing is that the user interface can be decoupled from the functionality of the applications (i.e., objects). For example, HTML code can be used as a front end to the Java applets. This gives the developer complete control over which applets are included on an HTML page and how this functionality is presented to the user.

Let's see what happens when you use the order-entry transaction example to check prices and availability. The order-entry server receives query input from the Java client applet and transmits a Java applet to the inventory server to search its database for the appropriate information. Note that this processing flow is significantly different from that of traditional client/server applications, where the client makes calls directly to remote procedures residing on one or more servers.



A Java-enabled Order Entry Application can use Java scripts to string applets together to create a full-blown business application.

After checking pricing and availability, a credit check could be performed by sending another Java applet to the accounts-receivable server. Next, Java applets actually commit the order and send transaction confirmation data to the client along with a Java applet to display it. Finally, the order is scheduled.

As this example shows, network-centric computing is both simpler and far more complex than traditional desktop-centric C/S computing.

It's simpler because it makes fewer demands on client workstations. Since all executable software is dynamically transmitted to the client, problems related to erroneously configured and out-of-date software versions virtually disappear.

Development shops are also relieved of the burden of creating and maintaining different versions of software for different platforms. Thus, Java based network-centric applications will allow organizations to build and customize business applications that will reach a wider audience much more quickly and with fewer support issues. In this case, Java truly gives you a "write-once, run-anywhere" capability.

But network-centric computing is also more complex because it must locate and then successfully transmit and launch several applets across several different servers and nodes across the network. This added complexity, however, does not diminish Java's ability to provide the traditional advantages of client/server systems, it merely enhances the application developer's ability to distribute processing across a network.

While we wouldn't recommended that organizations consider discarding their existing investments in desktop-centric C/S systems, it is likely that by using these new network-centric capabilities business organizations will benefit greatly by having a more flexible and, hence, more adaptable application architecture.

JIM FLYNN AND BILL CLARKE ARE THE PRINCIPALS OF @WORK TECHNOLOGIES, A LEADING JAVA DEVELOPMENT SHOP AND WEB SITE CONTENT PROVIDER IN NEW YORK CITY. THEY CAN BE REACHED THROUGH E-MAIL AT jflynn@worktechs.com AND wdclarke@worktechs. com, RESPECTIVELY. THEIR WEB SITE CAN BE FOUND AT http://www.work techs.com.

What Is This Thing Called JAVA?

JAVA IS A LOT OF THINGS TO A LOT OF PEOPLE. HERE'S A DOZEN OF ITS MOST IMPORTANT FEATURES, ACCORDING TO THE PEOPLE WHO INVENTED IT.

BY ARTHUR VAN HOFF, SAMI SHAIO, & ORCA STARBUCK

1. JAVA IS SIMPLE

Java is very similar to C++, but it's much simpler. All of the features of high-level programming languages that are not absolutely necessary have been left out. For example, Java has no operator overloading, header files, preprocessor, pointer arithmetic, structures, unions, multidimensional arrays, templates, or implicit type conversion. If you know a little C, C++, or Pascal, you'll be hacking in Java in no time. Here's a simple Java Hello World program:

```
1: public class HelloInternet {
2: public static void
main(String argv[ ]) {
3: System.out.println("Hello
Internet!");
4: }
5: }
```

2. JAVA IS OBJECT ORIENTED

Java is an object-oriented programming language. With the exception of simple types like numbers and Booleans, most things in Java are objects.

As with any object-oriented language, Java code is organized into classes. Each class defines a set of methods that define the behavior of an object. A class can inherit behaviors from another class. At the root of the class hierarchy is always class Object.

Java supports a single-inheritance class hierarchy. This means that each class can only inherit from one other class at a time. Some languages also allow multiple inheritance, but this can be confusing and make the language unnecessarily complicated. It's hard to imagine, for example, what an object would do that inherits behaviors from two totally different classes, say a Bike and a House.

Java also supports interfaces, which are abstract classes. This allows programmers to define methods for interfaces without immediately worrying how the methods will be implemented. A class can implement multiple interfaces. This has many of the advantages of true multiple inheritance without a lot of the problems. An object can also implement any number of interfaces. The Java interfaces are very similar to IDL interfaces. It's easy to build an IDL-to-Java compiler, which means that Java can be used in the CORBA object system to build distributed object systems. This compatibility is important because both IDL interfaces and the CORBA object system are used in many computer systems.

3. JAVA IS STATICALLY TYPED

In a Java program, the type of the objects (numbers, characters, arrays, etc.) that are used must be defined. This helps programmers find potential problems much sooner because type errors can be detected when a program is compiled.

However, all objects in the Java system also have a dynamic type. It's always possible to ask an object for its dynamic type, so a programmer can write programs that do different things for objects of different types.

4. JAVA IS COMPILED

When running a Java program, it is first compiled to bytecodes. Bytecodes are very similar to machine instructions, so Java programs can be very efficient. However, bytecodes are not specific to a particular machine, so Java programs can be executed on lots of different computers without recompiling the programs.

Java source programs are compiled to class files, which contain the bytecode representation of the program. In a Java class file, all references to methods and instance variables are made by name and are resolved when the code is first executed. This makes the code more general and less susceptible to changes, but still efficient.

5. JAVA IS ARCHITECTURE NEUTRAL

The Java language is the same on every computer. For example, simple types don't vary: an integer is always 32 bits and a long is always 64 bits. Surprisingly, this is not true for modern programming languages such as C and C++. Because these languages are so loosely defined, each compiler and development environment is slightly different, which makes porting a nightmare. Porting Java programs is easy, and it isn't necessary to recompile them.

6. JAVA IS ROBUST

Java programs can't cause a computer to crash. The Java system carefully checks each access to memory and makes sure it's legal and won't cause any problems.

But even Java programs can have bugs. If something unexpected happens, the program doesn't crash, but an exception is thrown. The program will find these exceptions and deal with them.

Traditional programs can access all of your computer's memory. A program can (unintentionally) change any value in memory, which can cause problems. Java programs can only access those parts of memory that they are allowed to access, so a Java program cannot change a value that it's not supposed to change.

7. JAVA IS SMALL

Because Java was designed to run on small computers, the Java system is relatively small, for a programming language. It can run efficiently on PCs with 4MB of RAM or more. The Java interpreter takes up only a few hundred kilobytes. The interpreter is responsible for Java's platform independence and portability.

Because the Java runtime is small, it is ideal for computers with very little memory such as the Internet-based PC [see "The Java-based Internet PC," page 33], as well as TVs, toasters, telephones, and home computers.

8. JAVA IS MULTITHREADED

A Java program can have more than one thread of execution. For example, it could perform some lengthy computation in one thread, while other threads interact with the user. So users do not have to stop working to wait for Java programs to complete lengthy operations.

Programming in a multithreaded environment is usually difficult because many things can happen at the same time. Java, however, provides easy-to-use features for synchronization that make programming easier.

Java threads are usually mapped onto real operating system threads if the underlying operating system supports this action. Thus, applications written in Java are said to be "MP-hot," which means they will benefit if they are executed on a multiprocessor machine.

9. JAVA IS GARBAGE COLLECTED

Programmers who write software in C and C++ have to keep careful track of each chunk of memory used. When a chunk is no longer used, they have to make sure the program frees it so that it can be used again. In large projects, this can become difficult, and it is often a source of bugs and memory leaks.

In Java, programmers don't have to worry about memory management. The Java system has a built-in program called the "garbage collector" that scans memory and automatically frees any memory chunk that's no longer in use.

10. JAVA IS FAST

Java is a lot more efficient than typical scripting languages, but it's about 20 times slower than C. This is acceptable for most applications.

In the near future, code generators will be available that will make Java programs nearly as fast as programs written in C or C++.

11. JAVA IS SECURE

Java programs have no pointers, and bytecode programs like Java are strongly typed, so it's possible to verify a Java program before executing it. A verified Java program is guaranteed not to break any of the Java language constraints and can be safely executed. Java bytecode verification is used by Web browsers to be sure applets don't contain viruses.

12. JAVA IS EXTENSIBLE

It's possible to interface Java programs to existing software libraries written in other languages. Because Java data structures are very similar to C's data structures and types, this is relatively easy. The biggest problem is that few existing libraries are multithreaded.

A Java program can declare certain methods to be native. These native methods are then mapped to functions defined in software libraries that are dynamically linked into the virtual machine.

EXCERPTED FROM *Hooked On Java:* BY ARTHUR VAN HOFF, SAMI SHAIO, AND ORCA STARBUCK (READING, MASS.: ADDISON-WESLEY, ©1995).

Yes, JAVA'S Secure. Here's Why

SECURE TRANSACTIONS ON THE INTERNET? YUP. HERE, TODAY, NOW. JUST RUN JAVA-COMPILED BYTECODE ON A JAVA VIRTUAL MACHINE, AND YOU CAN KEEP THE CYBERPUNKS OFF THE INTERNET.

BY LAURA LEMAY & CHARLES PERKINS

A GOOD RULE OF SAFE behavior on the Internet: Don't download any executable code unless you get it from someone (or some company) you know and trust.

Fortunately, Java allows you to relax that rule. You can run Java applets from anyone, anywhere, in complete safety.

Java's powerful security mechanisms act at four different levels of the system architecture. First, the Java language itself was designed to be safe, and the Java compiler ensures that source code doesn't violate these safety rules. Second, all bytecodes executed by the Java runtime engine are screened to be sure that they also obey these rules. (This layer guards against having an altered compiler produce code that violates the safety rules.) Third, the class loader ensures that classes don't violate name space or access restrictions when they are loaded into the system. Finally, API-specific security prevents applets from doing destructive things. This final layer depends on the security and integrity guarantees from the other three layers.

Here's how each layer works:

THE LANGUAGE AND THE COMPILER

The Java language and its compiler are the first line of defense. Java was designed to be a safe language.

Most other C-like languages have facilities to control access to objects but also have ways to forge access to objects (or to parts of objects) usually by (mis)using pointers. This introduces two fatal security flaws to any system built on these languages. One is that no object can protect itself from outside modification, duplication, or spoofing. Another is that a language with powerful pointers is more likely to have serious bugs that compromise security. These pointer bugs, where a runaway pointer starts modifying some other object's memory, were responsible for most of the public (and not-so-public) security problems on the Internet this past decade.

The language definition and the compilers that enforce it create a powerful barrier to any nasty Java programmer.

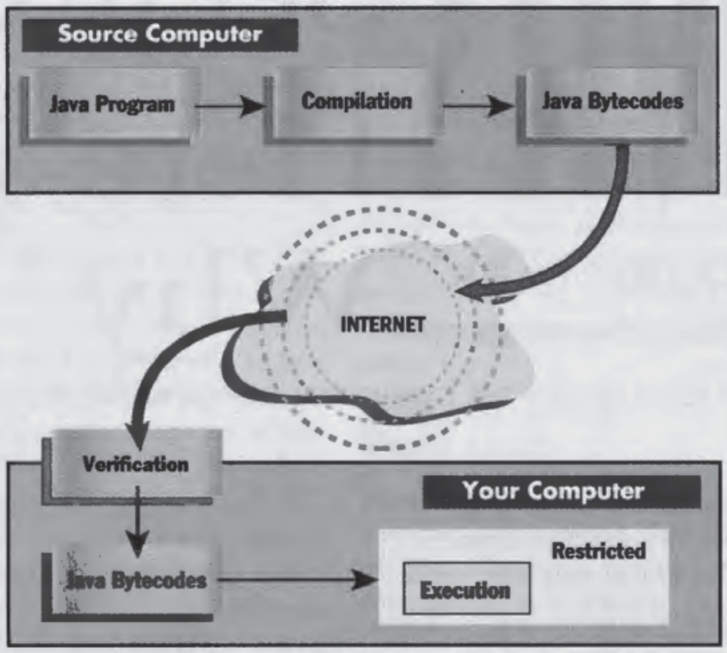
Java eliminates these threats in one stroke by eliminating pointers from the language altogether. There are still pointers of a kind–object references–but these are carefully controlled to be safe. In addition, powerful new array facilities in Java not only help to offset the loss of pointers but add additional safety by strictly enforcing array bounds, catching more bugs for the programmer (bugs that in other languages might lead to unexpected, and thus bad-guy-exploitable, problems).

The language definition and the compilers that enforce it create a powerful barrier to any nasty Java programmer.

VERIFYING THE BYTECODES

What if that nasty programmer gets a little more determined and rewrites the Java compiler to suit his or her nefarious purposes? The Java runtime engine, getting the lion's share of its bytecodes from the Net, can never tell whether those bytecodes were generated by a trustworthy compiler. Therefore, it must verify that they meet all the safety requirements.

The Java Verifier in Action



Before running any bytecodes, the runtime subjects them to a rigorous series of tests that vary in complexity from simple format checks all the way up to running a theorem prover. These tests verify that the bytecodes do not forge pointers, violate access restrictions, access objects as other than what they are (InputStreams are always used as InputStreams, and never as anything else), call methods with inappropriate argument values or types, or overflow the stack.

EXTRA TYPE INFORMATION AND REQUIREMENTS

Java bytecodes encode more type information than is strictly necessary for the interpreter. Even though, for example, the *aload* and *iload* opcodes do exactly the same thing, *aload* is always used to load an object reference and *iload* used to load an integer. Some bytecodes (such as *getfield*) include a symbol table reference–and that symbol table has even more type information. This extra type information allows the runtime system to guarantee that Java objects and data aren't illegally manipulated.

Conceptually, before and after each bytecode is executed, every slot in the stack and every local variable has some type. This collection of type information–all of the slots and local variables–is called the "type state" of the execution environment. An important requirement of the Java type state is that it must be determinable statically by induction–that is, before any program code is executed. As a result, as the runtime system reads the bytecodes, each is required to have the following inductive property: Given only the type state before the execution of the bytecode, the type state afterward must be fully determined.

The Verifer is the crucial piece of Java's security, and it depends on your having a correctly implemented runtime system.

Bytecode-based languages like Smalltalk and Postscript do not have this restriction. Their more dynamic type behavior does create additional flexibility, but Java needs to provide a secure execution environment.

THE VERIFIER

Bytecodes are checked for compliance with all of these requirements, using the extra type information in a .class file, by a part of the runtime called the Verifier. It examines each bytecode in turn, constructing the full type state as it goes, and verifies that all of the types of parameters, arguments, and results are correct. Thus, the Verifier acts as a gatekeeper to your runtime environment, letting in only those bytecodes that pass muster.

Warning: The Verifier is the crucial piece of Java's security, and it depends on your having a correctly implemented (no bugs, intentional or otherwise) runtime system. As of this writing, only Sun and Netscape are producing Java runtimes, and they are secure. In the future, however, you should be careful when downloading or buying another companies' (or individuals') version of the Java runtime environment. Eventually, Sun will implement validation suites for runtimes, compilers, and so forth to be sure that they are safe and correct. In the meantime, caveat emptor! Your runtime is the base on which all of the rest of Java's security is built, so make sure it is a good, solid, secure base.

When bytecodes have passed the Verifier, they are guaranteed not to:

- Cause any operand stack under- or overflows;
- Use parameter, argument, or return types incorrectly;
- Illegally convert data from one type to another (from an integer to a pointer, for example);
- Access any object's fields illegally

(that is, the Verifier checks that the rules for public, private, package, and protected are obeyed).

As an added bonus, because the interpreter can now count on all of these facts being true, it can run much faster than before. All of the required checks for safety have been done up front, so it can run at full throttle.

THE CLASS LOADER

The class loader is another kind of gatekeeper, albeit a higher level one. The Verifier was the security of last resort. The class loader is the security of first resort.

When a new class is loaded into the system, it must come from one of several different realms. In the current release, there are three possible realms: your local computer, the firewall-guarded local network on which your computer is located, and the Internet (the global Net). Each of these realms is treated differently by the class loader.

Actually, there can be as many realms as your desired level of security (or paranoia) requires. This is because the class loader is under your control. As a programmer, you can make your own class loader that implements your own peculiar brand of security.

As a user, you can tell your Java-aware browser, or Java system, what realm of security (of the three) you'd like it to implement for you right now or from now on.

As a system administrator, you can set up global security policies to help guide your users to not give away the store (that is, set all their preferences to be unrestricted, promiscuous, "hurt me please!").

In particular, the class loader never allows a class from a less-protected realm to replace a class from a more-protected realm. The file system's I/O primitives, about which you should be very worried, are all defined in a local Java class, which means that they all live in the local-computer realm. Thus, no class from outside your computer (from either the supposedly trustworthy local network or from the Internet) can take the place of these classes and spoof Java code into using nasty versions of these primitives. In addition, classes in one realm cannot call upon the methods of classes in other realms, unless those classes have explicitly declared those methods public. This implies that classes from other than your local computer cannot even see the file system I/O methods, much less call them, unless you or the system wants them to.

Caveat emptor! Your runtime is the base on which all of Java's security is built, so make sure it is a good, solid, secure base.

In addition, every new applet loaded from the network is placed into a separate package-like namespace. This means that applets are protected even from each other. No applet can access another's methods (or variables) without its cooperation. Applets from inside the firewall can even be treated differently from those outside the firewall.

In practice, it's all a bit more complex than this. In the current release, an applet is in a package namespace along with any other applets from that source. This source, or origin, is most often a host (domain name) on the Internet. Depending on where the source is located, outside the firewall (or inside), further restrictions may apply (or be removed entirely). This model is likely to be extended in future releases of Java, providing an even finer degree of control over which classes get to do what.

Consider the full lifetime of a Java method. It starts as source code on some computer, is compiled into bytecodes on some (possibly different) computer, and can then travel (as a .class file) into any file system or network anywhere in the world. When you run an applet in a Java-aware browser (or download a class and run it by hand using Java), the method's bytecodes are extracted from its .class file and carefully looked over by the Verifier. Once they are declared safe, the interpreter can execute them for you (or a code generator can generate native binary code for them using either the just-in-time compiler or the java2c bytecode compiler, and then run that native code directly).

At each stage, more and more security is added. The final level of that security is the Java class library itself, which has several carefully designed classes and APIs that add the final touches to the security of the system.

One final note about security. Despite the best efforts of the Java team, there is always a trade-off between useful functionality and absolute security. For example, Java applets can create windows, an extremely useful capability, but a nasty applet could use this to spoof the user into typing private password information, by showing a familiar program (or operating system) window and then asking an expected, legitimate-looking question in it. (The beta release adds a special banner to applet-created windows in an attempt to solve this problem.)

Flexibility and security can't both be maximized. Thus far on the Net, people have chosen maximum flexibility and have lived with the minimal security the Net now provides. Let's hope that Java can help tip the scales a bit, enabling better security, while sacrificing only a minimal amount of flexibility.