Chapter 6. Getting Started with Java

Topics in This Chapter

- Unique features of Java: What's so unusual, and why everyone is so excited about it
- Java myths: Separating hype from reality
- The evolution of Java: Past and current Java versions
- Ready, set: The Java software and documentation you'll need
- Go: How to compile and run a Java program
- Appetizers: Some simple Java programs
- Java in the real world: Some sample Java applications

Java is a programming language that looks a lot like C++ and can be used for general-purpose applications, for embedding programs in WWW pages, and for enterprise programs on e-commerce sites. Programmers think it's cool; software managers think it's hot. In fact, two companies in particular, IBM and Oracle, love Java so much that the first step in their Enterprise Developer program is to become certified in Java (see http://www-4.ibm.com/software/ad/certify/adedserv.html and http://education.oracle.com/certification/javatrack.html).

Why is Java so hot? Here are a few reasons that we'll discuss in Section 6.1.

- Java is Web-enabled and network savvy
- Java is cross-platform
- Java is simple
- Java is object oriented
- Java is rich with powerful standard libraries

Although Java can do Windows and take out the garbage, some Java advocates would have us believe that Java is the One True Programming Language to which all hitherto unenlightened programmers will convert for all applications. On the other hand, some of the supposed drawbacks to Java are imagined or exaggerated as well. In Section 6.2 we debunk these common myths:

- Java is only for the Web

- Java is cross-platform
- Java is simple
- Java is object oriented (the one true way of programming)
- Java is the programming language for all software development

If you're not sure which version of Java is for you, Section 6.3 summarizes the differences among the available releases. Now, if somehow you've been in a time capsule and don't already have Java on your computer, we show you in Section 6.4 where to get all the software and documentation you need. Finally, Section 6.5 shows some simple, ready-to-run programs to get you started.

If you are wondering what Java stands for, it is not an acronym at all; the word was chosen because of the use of "java" as American slang for "coffee." This choice came about after Sun had to change the original name (Oak) because of a conflict with an existing name. The name Java is meant to imply something exciting and hip. Digging a little deeper: coffee became known as "java" because of coffee imports from Indonesia, where the main population lives on the island of Java. But where did the island get its name? Turns out that Java came from Yava, which meant "rice" in a dialect of the eighth century, at which time Java was known as Yava Dwipa, or Rice Island.

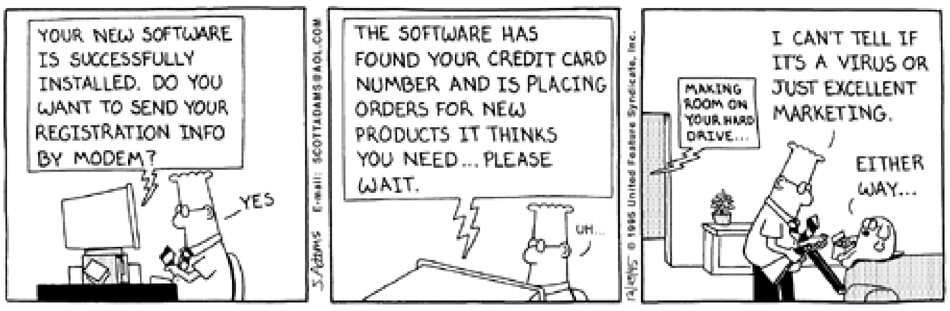
6.1 Unique Features of Java

Actually, many of the capabilities described in this section are *not* truly one of a kind, just unique in the experience of most developers. Java is an excellent language, but not, as some proponents claim, a brilliant breakthrough packed with ideas that have never been seen before. The vast majority of Java language features are already available in other languages. So why all the excitement about Java? What Java has done that few other languages have been able to do is to combine standard capabilities (C/C++ syntax), powerful features of niche-market languages (automatic memory management, bytecode interpreters), and key APIs for enterprise development. The following sections highlight some of the most important characteristics of Java.

Java Is Web-Enabled and Network Savvy

The World Wide Web helped catapult Java to its current prominent position as an Internet programming language. If you're writing applications that need to run on the Web, access Internet resources, or simply talk to other programs on the network, Java has numerous facilities to make your life easier.

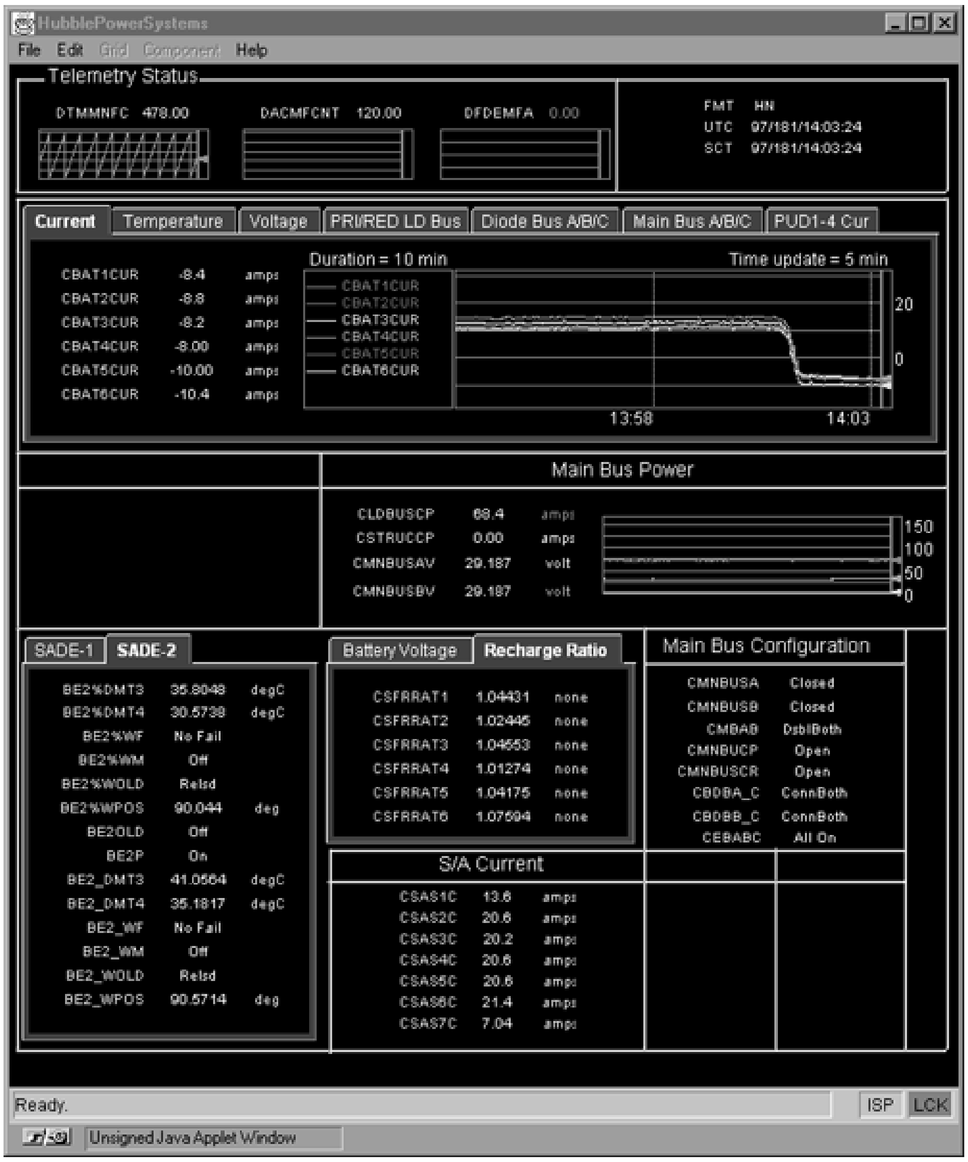
> **Java Enforces Safety** Because Java checks array bounds, forbids direct manipulation of memory addresses, and enforces datatypes, Java programs cannot access arbitrary memory locations. Thus, before execution of any line of code, a security manager can reliably analyze what operations the program can legally perform. This analysis permits a restricted class of Java programs known as "applets" to be run in your Web browser without danger of introducing viruses, finding and reporting on private information about your system, erasing your disk, snooping behind your corporate firewall, or starting up programs like Doom just when your boss is entering your office.


YOUR NEW SOFTWARE IS SUCCESSFULLY INSTALLED. DO YOU WANT TO SEND YOUR REGISTRATION INFO BY MODEM?
YES
THE SOFTWARE HAS FOUND YOUR CREDIT CARD NUMBER AND IS PLACING ORDERS FOR NEW PRODUCTS IT THINKS YOU NEED... PLEASE WAIT.
UH...
MAKING ROOM ON YOUR HARD DRIVE...
I CAN'T TELL IF IT'S A VIRUS OR JUST EXCELLENT MARKETING.
EITHER WAY...
S. Adams E-mail: SCOTTADAMS@AOL.COM
© 1995 United Feature Syndicate, Inc.

DILBERT © UFS. Reprinted with permission.

The Web Can Deliver Software Java applets are supported in most versions of Netscape and Internet Explorer, and, therefore, applets run on nearly every operating system on the market and are available to virtually anyone on the Internet. This capability opens up a whole new way of viewing the WWW and a browser: as a medium for software delivery and execution, not just as a medium for document delivery and display. If you have an application that you update frequently, you no longer need your users to reinstall the latest version every time you make a change. In fact, your users don't need to install anything at all; all they have to do is keep a bookmark to your applet in the Web browser they already have on their system. This shifts the burden of software installation and maintenance from the user to the developer, who can control versions at a single centralized location. For example, the Hubble Control Center System, shown in Figure 6-1, is accessed in a variety of locations over the Web, providing up-to-the-minute status on the Hubble Space Telescope.

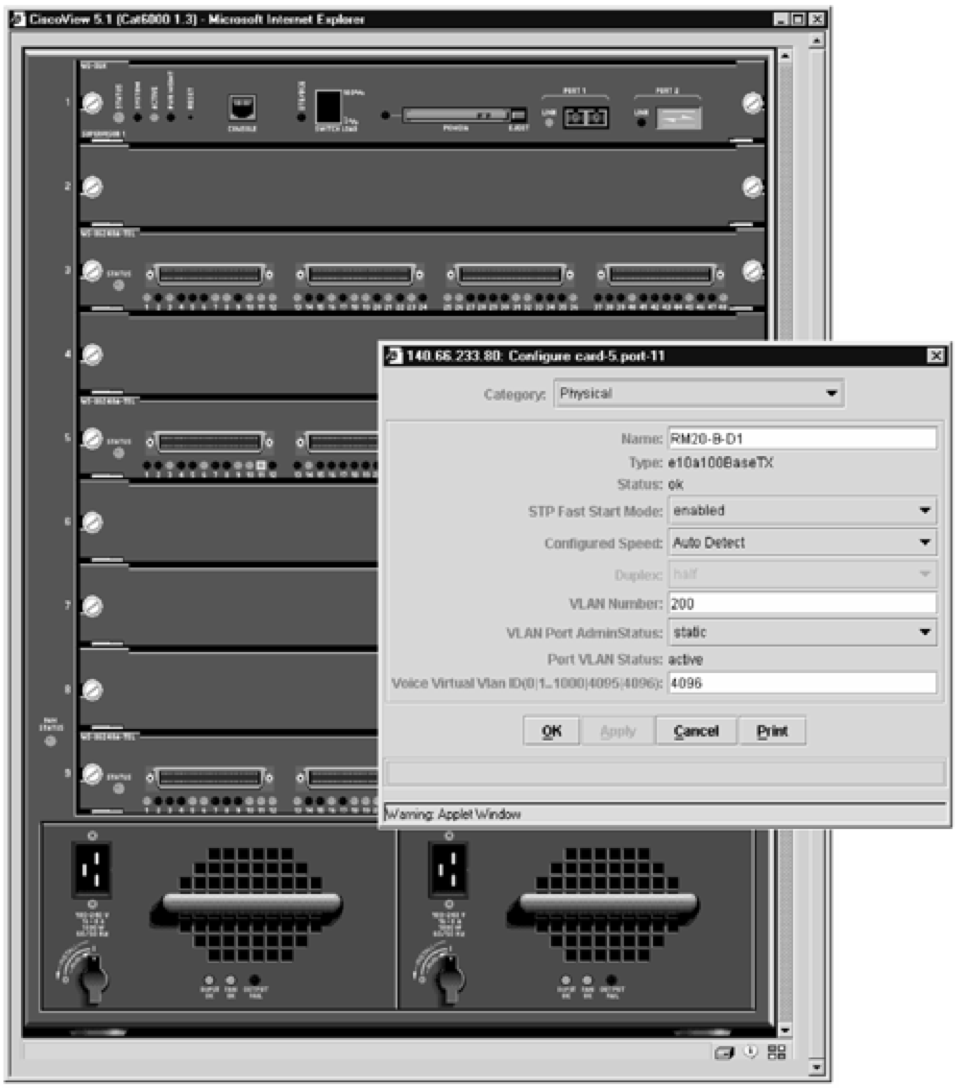
Figure 6-1. Control Center System applet for Hubble Space Telescope, developed at NASA.



```
DTMMNFC   478.00     DACMFCNT   120.00     DFDEMFA   0.00

FMT   HN
UTC   97/181/14:03:24
SCT   97/181/14:03:24

CBAT1CUR    -8.4     amps
CBAT2CUR    -8.8     amps
CBAT3CUR    -8.2     amps
CBAT4CUR    -8.00    amps
CBAT5CUR    -10.00   amps
CBAT6CUR    -10.4    amps

Main Bus Power
CLDBUSCP    68.4     amps
CSTRUCCP    0.00     amps
CMNBUSAV    29.187   volt
CMNBUSBV    29.187   volt

SADE-2
BE2%DMT3    35.8048  degC
BE2%DMT4    30.5738  degC
BE2%WF      No Fail
BE2%WM      Off
BE2%WOLD    Relsd
BE2%WPOS    90.044   deg
BE2OLD      Off
BE2P        On
BE2_DMT3    41.0564  degC
BE2_DMT4    35.1817  degC
BE2_WF      No Fail
BE2_WM      Off
BE2_WOLD    Relsd
BE2_WPOS    90.5714  deg

Recharge Ratio
CSFRRAT1    1.04431  none
CSFRRAT2    1.02445  none
CSFRRAT3    1.04553  none
CSFRRAT4    1.01274  none
CSFRRAT5    1.04175  none
CSFRRAT6    1.07594  none

Main Bus Configuration
CMNBUSA     Closed
CMNBUSB     Closed
CMBAB       DsblBoth
CMNBUCP     Open
CMNBUSCR    Open
CBDBA_C     ConnBoth
CBDBB_C     ConnBoth
CEBABC      All On

S/A Current
CSAS1C      13.6     amps
CSAS2C      20.6     amps
CSAS3C      20.2     amps
CSAS4C      20.6     amps
CSAS5C      20.6     amps
CSAS6C      21.4     amps
CSAS7C      7.04     amps
```

Java's Network Library Is Easy to Use Java's networking library is used exactly the same way on all operating systems. Ordinary mortals can actually use the network library, a welcome change from other languages where you leave such magic to the local wizards who are probably on another project just when you need them the most. Compared to, for instance, Berkeley sockets or the POSIX Transport Layer Interface, creating clients or servers is positively a joy in Java. Furthermore, Java already understands the HTTP protocol, letting you retrieve files on the Web and communicate with HTTP servers without even dealing directly with sockets. For example, Figure 6-2 shows a Web-based interface (applet) for configuring ports on a Cisco Catalyst 6000 (Ethernet switch). The HTTP service running on the Catalyst switch permits configuration through a browser instead of entering command-line changes over a telnet session.

Figure 6-2. Cisco OpenView applet for configuring a Catalyst 6000 Ethernet switch. [This material has been reproduced by Prentice Hall with the permission of Cisco Systems, Inc. © 2001 Cisco Systems, Inc. All rights reserved.]



140.66.233.80: Configure card-5.port-11

Category: Physical

Name: RM20-B-D1
Type: e10a100BaseTX
Status: ok
STP Fast Start Mode: enabled
Configured Speed: Auto Detect
Duplex: half
VLAN Number: 200
VLAN Port AdminStatus: static
Port VLAN Status: active
Voice Virtual Vlan ID(0|1..1000|4095|4096): 4096

Warning: Applet Window

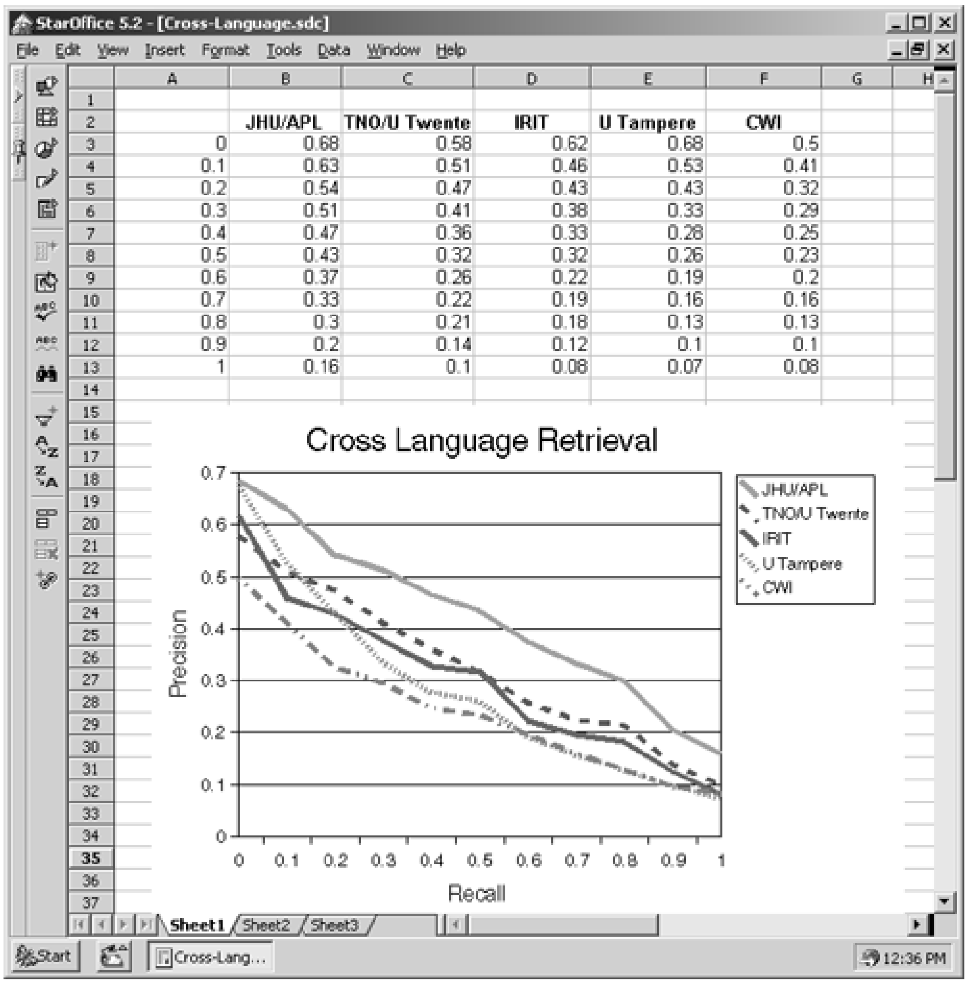
Java Provides Complete Enterprise Development Since 1995, Sun has introduced numerous libraries and extensions to Java in support of enterprise solutions. Java supports Remote Method Invocation (RMI) that lets you invoke methods in objects on remote platforms, allowing you to pass arbitrary Java objects back and forth across the network. Java servlets let you write server-side applications and create session objects to track users across multiple Web pages. JDBC lets your applications interface with multiple database vendors in a standard manner and lets your applets or servlets bypass older CGI interfaces and talk directly to a database. Enterprise JavaBeans components, accessible from servlets or JavaServer pages, allow you to store business rules in a multi-tier architecture.

Java Is Cross-Platform

Java is designed to be portable, and Java programs developed on one platform can often run unchanged on many other computer systems. Why? Well, a number of reasons exist, but the three most important characteristics that make Java portable are outlined here.

Java Compiles to Machine-Independent Bytecode Java is typically compiled and executed in a two-step process. In the first step, Java source code is compiled to "bytecode"—assembly language for an idealized Java Virtual Machine (JVM). In the second step, this bytecode is executed by a run-time system. This run-time system can either be an interpreter (an emulator for the JVM) or a Just In Time (JIT) compiler that first compiles the bytecode to native code, then executes the result. The beauty of this process is that the two steps can be performed on totally separate platforms. The source can be compiled on a Windows 2000 machine with Borland's compiler, then the result can be executed on a Macintosh through Apple's run-time system, or on Solaris with Sun's software. For example, Figure 6-3 presents StarOffice™, a cross-platform, document processing suite that runs on Windows, Solaris, and Linux. In addition, most modern Web browsers include a JVM, letting Web page developers compile applets and attach the resultant bytecode to Web pages for execution on a variety of platforms.

Figure 6-3. StarOffice 5.2, a cross-platform suite for document processing written completely in Java. StarOffice™ is a registered trademark of Sun Microsystems, Inc.



	JHU/APL	TNO/U Twente	IRIT	U Tampere	CWI
0	0.68	0.58	0.62	0.68	0.5
0.1	0.63	0.51	0.46	0.53	0.41
0.2	0.54	0.47	0.43	0.43	0.32
0.3	0.51	0.41	0.38	0.33	0.29
0.4	0.47	0.36	0.33	0.28	0.25
0.5	0.43	0.32	0.32	0.26	0.23
0.6	0.37	0.26	0.22	0.19	0.2
0.7	0.33	0.22	0.19	0.16	0.16
0.8	0.3	0.21	0.18	0.13	0.13
0.9	0.2	0.14	0.12	0.1	0.1
1	0.16	0.1	0.08	0.07	0.08

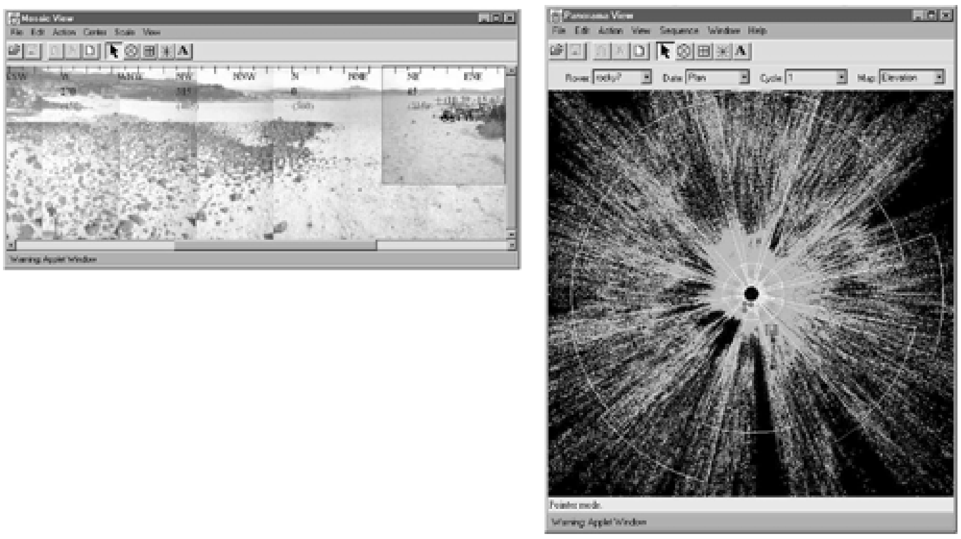
Java Offers a Portable Graphics Library In many software systems, the biggest hindrance to portability was the user interface. Interfaces were typically developed with the native windowing system rather than a cross-platform graphics toolkit because this was the most convenient and widely available option. However, this approach often meant that distribution on a different operating system required a complete rewrite of the GUI. The Java developers realized that a truly portable language would require a standard graphics library and, in the original release of Java, Sun

Microsystems introduced the Abstract Window Toolkit (AWT) for platform-independent GUI development. The AWT provides a standard set of graphical controls (buttons, lists, combo boxes, check boxes, textfields, etc.) for development of stand-alone application and applet-based GUIs that are supported in nearly all browsers.

Later, in the Java 2 Platform release, Sun introduced Swing—a richer, more robust, graphical package for creating professional user interfaces. Swing is the preferred library for developing graphical programs, unless you are writing applets for Web pages. Among the various versions of Netscape and Internet Explorer, Netscape 6 is the only browser that currently supports Swing without a plug-in.

Figure 6-4 shows the applet-based GUI created when NASA was faced with the problem of making Pathfinder data available to a huge number of users on different platforms.

Figure 6-4. Jet Propulsion Lab's Web Interface for Telescience, used for worldwide viewing of Mars Pathfinder data.



Java Avoids Hard-to-Port Constructs The Java specification defines the size of primitive datatypes such as `int`s, `boolean`s, and `double`s, unlike other languages that allow the size to vary among implementations. For objects, Java programs can't imprudently depend on implementation-specific details such as the amount of memory an object consumes (Java has no need for the equivalent to the C/C++ `sizeof` operator), the internals of how fields or functions are laid out within an object, or the like. Java even avoids reference to the local file system when specifying which classes your program requires, using operating-system-neutral class and package names instead.

Java Is Simple

Java started with familiar C++ syntax but cleaned up many of the complicated syntactic features. In addition, header files are never needed, makefiles are usually not necessary, and the networking process is easier, not to mention numerous other improvements. In addition to a long list of similar minor features, Java has two major features that simplify life for the programmer: automatic memory management and simplified pointer handling.

Java Automatically Manages Memory The Java programmer is freed from the time-consuming and error-prone process of manually allocating and deallocating memory for objects. Instead, an automatic system, known as a *garbage collector,* doles out memory when it is needed and reclaims memory from objects that can no longer be accessed. Poof! In one fell swoop, Java has eliminated dangling pointers (references to memory that has been recycled) and memory leaks (inaccessible memory that is never reclaimed)—two problems that often account for half of the development time in large systems programmed in languages with manual memory management.

Java Simplifies Pointer Handling When you pass an object (i.e., any nonprimitive data type) to a function in Java, the system actually passes a pointer or "reference" to the object. That is, the entire object is not copied onto the run-time stack, only the reference is copied. All the details are hidden from the user, who can simply view "the object" as being passed to the function. You do not need to explicitly reference or dereference the pointer; pointer arithmetic is unnecessary (banned, in fact), and the whole process is considerably simpler. Automated memory management lets the programmer stop thinking in terms of pointers altogether if desired, while still making it easy to implement data structures that depend on pointers, such as linked lists and trees. Although at first this seems strange to the C or C++ programmer, it is the way things have worked for decades in languages like Smalltalk and Lisp.

Java Is Object Oriented

Java is pervasively and consistently object oriented. "Object obsessed," some people would say.

All Functions Are Associated with Objects In many other object-oriented languages, there are "normal" functions that are independent of objects, as well as "methods" or "member functions" that are associated with objects. Java, however, is like Smalltalk in this regard, where methods are the *only* type of allowable function.

Almost All Datatypes Are Objects In some object-oriented languages, a distinction exists between regular datatypes and classes. Strings, arrays, structures, files, sockets, and other types might not be objects that can be processed in the same way as user-defined objects. In contrast, in Java all complex types are true objects, and every object has a common ancestor, the `Object` class, thus simplifying the creation of arrays or other collections of heterogeneous object types. Although a few "primitive" datatypes (`int`, `double`, `boolean`, `char`, among others) are kept distinct from objects for efficiency, there is a corresponding wrapper object for each of them (`Integer`, `Double`, `Boolean`, `Character`, and so forth). These wrapper objects can be obtained from the primitive type through simple conversion methods.

Java Is Rich with Powerful Standard Libraries

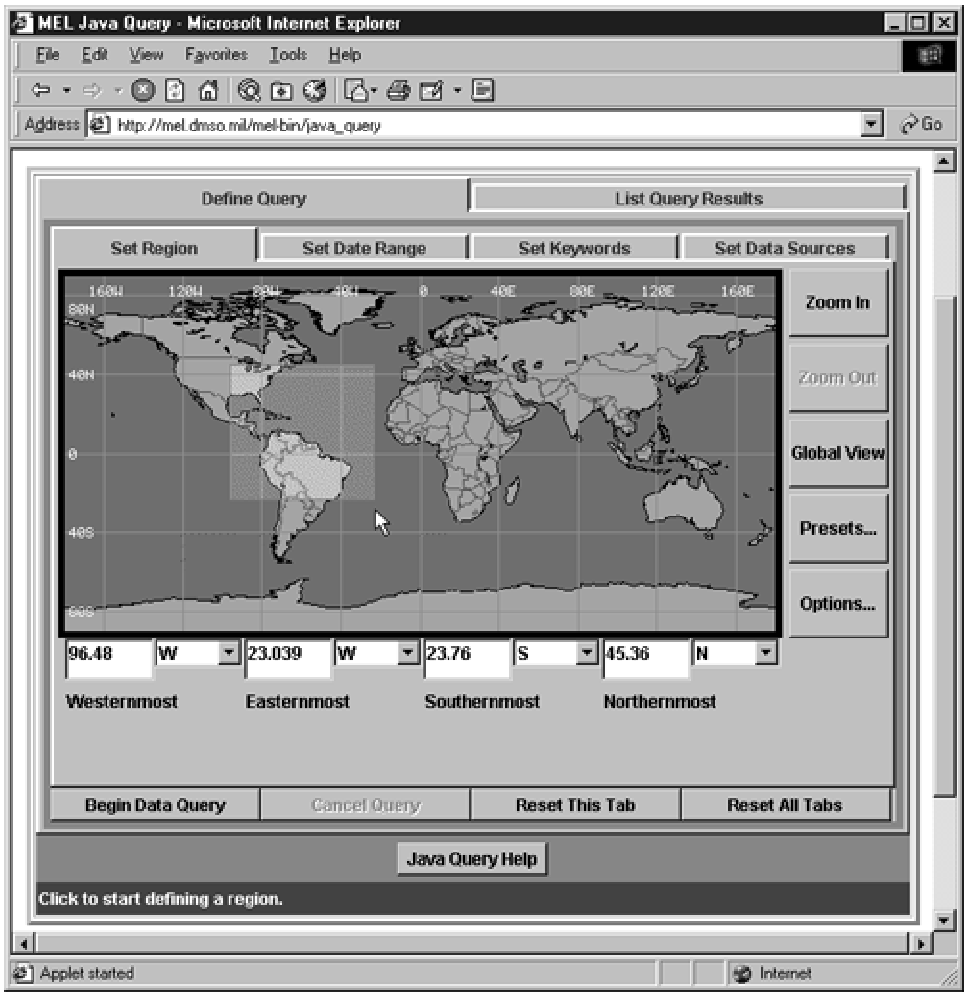
In addition to the graphics and client/server libraries already mentioned, Java has standard libraries for, to name a few, the following tasks:

- Building and using data structures
- Manipulating and parsing strings and streams
- Saving objects (even graphical ones) to disk and reassembling them later
- Using arbitrary-precision, fixed-point numbers
- Accessing files over the Internet

- Granting security privileges based on digital signatures
- Invoking remote Java objects
- Interfacing with relational databases
- Distributing computation among multiple threads of execution

As a result, you can write large applications completely in Java without recourse to libraries specific to a particular operating system. Figure 6-5 illustrates one such system.

Figure 6-5. The Java query interface to the Master Environmental Library (MEL) for geospatial data discovery and retrieval. A Java applet provides an interactive graphical specification of the region of interest. From Naim Alper and the MEL Project.



Westernmost		Easternmost		Southernmost		Northernmost	
96.48	W	23.039	W	23.76	S	45.36	N

Click to start defining a region.

Now, such a wide array of built-in capabilities is a mixed blessing. On the one hand, it provides a large set of standard, portable tools that you can pick through for your particular application. On the other hand, Java as a whole is so large that learning it seems intimidating. Fortunately, you don't have to learn Java all at once. Everyone will need to know how to construct and use objects (Chapter 7) and understand the core syntax (Chapter 8). Other than that, the pieces are fairly independent. Because Java loads classes dynamically, the size of the Java run-time environment and compiled classes is independent of the total language size. And you don't have to get a handle on everything before making good progress. If you're going to be writing a server that doesn't require a user interface, you can skip learning about the AWT (Chapter 13) and Swing (Chapters 14 and 15) until you actually need them. Similarly, you need not know anything about the networking library when writing a typical applet. Even so, no doubt you'll occasionally write some utility, only to discover later that it is already available in one of Java APIs. Okay, we admit that seems frustrating. But it's still better than using a bare-bones language where you *always* have to write every utility you need.

6.2 Myths About Java

When Java burst on the software scene in 1995, many misconceptions popped up. Because some of these misconceptions still persist, we'll try to refute the most prevalent ones here.

Java Is Only for the Web

When Java was first introduced, the majority of hype revolved around creating Java applets. Certainly, Java applets can run in Web pages; however, Java is a general-purpose programming language and does not require applets or the Internet. You could use Java to write a new driver for a color printer or even to write a numerical integration library. Java is not limited to the Internet. That's not to say that Java isn't the language of choice for Internet programming. With support for database connectivity, Java Servlets, Enterprise JavaBeans, and JavaMessaging, Java is the Internet language of choice! As more applications become distributed and more companies enter the e-commerce and business-to-business marketplace, Java will dominate Web programming. But don't get locked into thinking that Java is strictly for the Web.

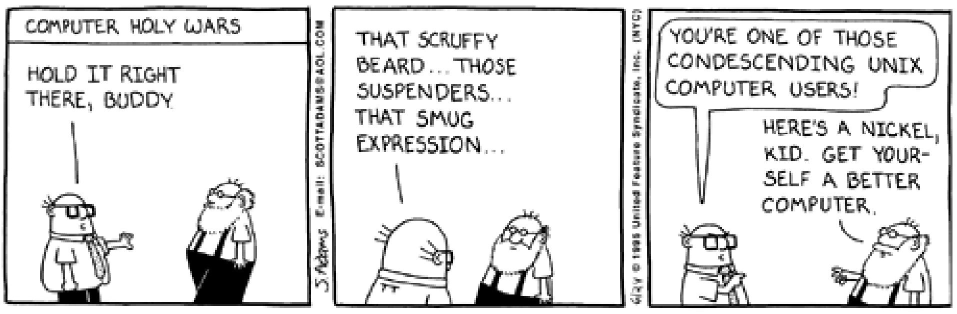
Java Is Cross-Platform

Yes, Java was designed with portability in mind and has succeeded in many ways. Many Java programs run unchanged on multiple operating systems. Nevertheless, some things can make some Java programs difficult or even impossible to port to other platforms. We describe the three most important ones here.

Java Programs Can Execute Local, Nonportable Programs The `exec` method of the `Runtime` class lets Java applications call local programs. The "native method" interface supports linking Java and C programs together. The first can make Java programs completely nonportable, and the second makes them only as portable as the associated C code. Now, it is hardly fair to say that Java is nonportable just because it connects to something that is nonportable. After all, that is not "pure" Java. True, but it is still worth remembering that not all "Java" programs are completely written in Java and that many do indeed connect to other components. Many programs, of course, can avoid accessing native code. But some programs need to format a disk, get a list of users currently logged in, or link with legacy C or C++ code, even if the programs need to sacrifice portability to do so. This behavior is not necessarily a bad thing; accessing local applications is sometimes more important than portability. The important thing is not to sacrifice portability *without knowing it,* and Java has done a good job of avoiding nasty surprises in this regard. Be wary of third-party libraries that don't make it clear which part is pure Java and which part is nonportable.

The Behavior of the Thread Scheduler Is Loosely Defined Java has one of the best threading libraries around. It lets you execute independent parts of your program in separate processes. However, to allow a compiler to use efficient mechanisms on a particular OS, some of the interactions among threads are only loosely specified. For instance, how long a given thread will run before being replaced by another is almost certain to vary from machine to machine and even from run to run on the same machine. In some sense, this nondeterminism is an advantage of using threads, but it can lead beginning programmers to inappropriately depend on certain behavior by the scheduler in order to operate correctly. If your program depends on thread priorities or the time interval in which threads are swapped in and out of the CPU, your program may not run the same on all platforms. Solaris, Windows NT, and Macintosh all use different priorities and algorithms for scheduling threads. So, when writing multithreaded programs using Java, you should be careful not to deliver your product before you thoroughly test the software on all platforms.

The Graphics Library Behaves Differently on Different Systems In the first release of Java, the development team decided that the graphic elements should adopt the look and feel of the local windowing system, rather than trying to have a consistent look and feel across platforms. In some sense, this was a wise approach, since experience has shown that some Macintosh users would rather die than use an application that looks like Windows 2000, no matter how much you preach that they should. But supporting a native look in the Abstract Window Toolkit came at a price: scrollbars, buttons, textfields, and the like can have slightly different sizes and behave slightly differently on different platforms. Some of these problems associated with component sizes on different operating systems can be minimized through the proper selection of layout managers. But, as Sun quickly realized, for Java to be accepted as a true cross-platform language for GUI development, the platform dependence of the component behavior would need to be resolved.


COMPUTER HOLY WARS
HOLD IT RIGHT THERE, BUDDY
THAT SCRUFFY BEARD... THOSE SUSPENDERS... THAT SMUG EXPRESSION...
YOU'RE ONE OF THOSE CONDESCENDING UNIX COMPUTER USERS!
HERE'S A NICKEL, KID. GET YOURSELF A BETTER COMPUTER.

DILBERT © UFS. Reprinted by permission.

In the Java 2 Platform, Sun introduced Swing, a new and complete graphical package that does not rely on the underlying operating system for rendering the components. In essence, a GUI written in Swing is truly platform independent. Unfortunately, if you intend to deliver your Swing GUI through a web-based applet, most browsers do not directly support the Swing components without a plug-in. In contrast, an applet written with AWT components is supported in nearly all available browsers.

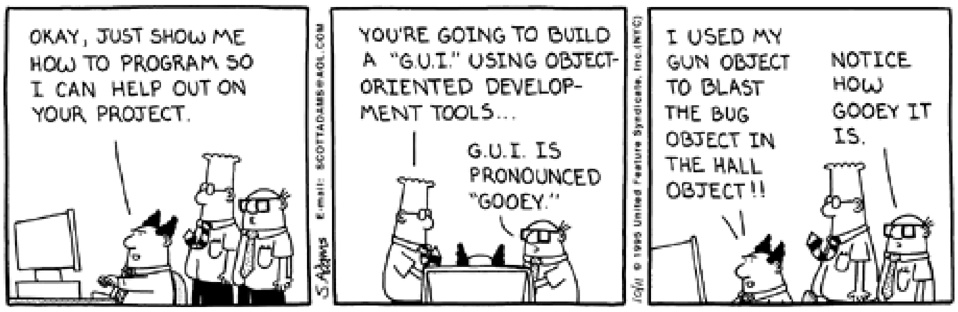
Java Is Simple

Simple? Ha. Who said it was simple? Oh, we did? Hmm.

The problem is that simplicity is a relative concept. Compared to some of the intricacies of C and C++, Java syntax seems positively streamlined. People who have fought with memory leaks and dangling pointers are relieved to let the Java garbage collector take over the battle. But Java is a full-blown programming language, and programming is anything but simple. So, for instance, the HTML developer will find Java programming a huge leap up in complexity. And given the ever-increasing number of standard and third-party libraries, it is becoming harder and harder to be knowledgeable in all of them.

Not only is programming hard, but programmers tend to continually push the envelope of what can reasonably be done. In many other languages, only the guru would attempt client-server or multithreaded applications, but Java makes these techniques accessible to "ordinary" programmers. This is a double-edged sword. On the one hand, developers can do useful things they couldn't do before. On the other hand, programs using these techniques can be very tricky to develop and debug.

So, Java programming is certainly not simple. Far from it. But Java is designed so that the various libraries and approaches can be learned in bite-sized pieces, and the relatively clean syntax and design make the learning process more palatable than in most other languages.


OKAY, JUST SHOW ME HOW TO PROGRAM SO I CAN HELP OUT ON YOUR PROJECT.
YOU'RE GOING TO BUILD A "G.U.I." USING OBJECT-ORIENTED DEVELOPMENT TOOLS...
G.U.I. IS PRONOUNCED "GOOEY."
I USED MY GUN OBJECT TO BLAST THE BUG OBJECT IN THE HALL OBJECT!!
NOTICE HOW GOOEY IT IS.

DILBERT © UFS. Reprinted by permission.

Java Is Object Oriented (the One True Way of Programming)

"It's not?" Blasphemy! you say. There are various ways to view programming. Some people take the religious view, where technical evangelists argue fervently to convert disciples of one software dogma to another. In some arenas, there *is* a single right answer. But we think the carpenter's model fits the software world better. Under this analogy, the various technical approaches are tools in the software developer's toolbox. Clearly, some tools are more broadly applicable than others, and certain tools are well suited to certain jobs. OOP is a useful and broadly applicable tool, and it should occupy a central place in the developer's toolkit. But functional programming, structured programming, rule-based programming, divide-and-conquer approaches, greedy algorithms, and the like are also useful tools, and the expert craftsman should be skilled with them as well. OOP is complementary to some of them, independent of others, and occasionally in conflict with some.

Choosing OOP as the underlying structure for Java was a wise choice, but once in a while the object-oriented viewpoint will get in the way. Rejecting a useful technique in Java simply because it "doesn't fit with the object-oriented philosophy" is, well, heresy.

Java Is the Programming Language for All Software Development

Java is a good general-purpose programming language. It is an excellent tool, perhaps even the best, for a number of jobs. But it's not the best tool for *every* job. Sometimes it will be more convenient to write a Unix utility in C, a Windows utility in Visual Basic, or a quick Web application in JavaScript or VBScript. To return to the carpenter analogy, the expert practitioner will be more successful knowing the strengths and weaknesses of various tools instead of using the same tool in all circumstances. On the other hand, portability and interoperability are important considerations, and heterogeneous systems tend to be less portable and interoperable than homogeneous ones. So sticking to Java even when a small piece of the system is easier in another language is sometimes preferable. Knowing where the balance lies requires experience.

6.3 Java Versions

Java 1.0 was first released by Sun Microsystems in 1995 and took off in early 1996 when

Netscape released Navigator version 2.0, the first widely used browser that supported Java applets. After a couple of bug fixes, Java 1.02 was released and is what most people mean when they say "Java 1.0." Later, in early 1997, Sun released Java 1.1, which contained a wide variety of enhancements and new features, including:

- A new event-handling model based on listeners
- Remote method invocation (RMI) and object serialization
- Support for inner and anonymous classes
- Arbitrary precision integers and floating-point numbers
- Java DataBase Connectivity (JDBC) API for connecting relations databases
- JavaBeans component architecture (Java's answer to ActiveX)
- Digitally signed applets to extended security privileges without resorting to the "all or nothing" model of browser plug-ins or ActiveX

Your best strategy when writing applets is to write code specific to the Java 1.1 API, since this version of Java is supported by Netscape 4.06 and later and Internet Explorer 4.0 and later. By choosing to use capabilities from later versions of Java, you run the risk of your applet not being supported by all customers that download your applet to their browser.

The Java 2 Platform, often identified by users as JDK 1.2 (Java Development Kit 1.2), was released in December of 1998. Significant changes introduced in JDK 1.2 include:

- Swing GUI components based on 100% Pure Java
- Java 2D for professional, high-quality, two-dimensional graphics and imaging
- The Collections Framework supporting advanced data structures like linked lists, trees, and sets
- Audio enhancements to support `.wav`, `.aiff`, `.au`, `.midi`, and `.rmf` file formats
- Printing of graphic objects
- Java IDL API, which adds CORBA capability to Java

In the Spring of 2000, Sun released JDK 1.3. In this version, minor fixes and enhancements were introduced throughout the API. The two most significant enhancements were:

- Java Naming and Directory Interface (JNDI)—a directory service for registering and looking up resources (objects)
- RMI-IIOP—a protocol to communicate with distributed clients that are written in CORBA-compliant language

Now, if those lists aren't daunting enough, JDK 1.2 and JDK 1.3 are really part of the Java 2 Platform, *Standard Edition.* At the same time the Standard Edition was released, Sun also introduced the Java 2 Platform, *Enterprise Edition,* for e-commerce solutions. The Enterprise Edition adds:

- Java Servlets and JavaServer Pages—Sun's answer to Microsoft Active Server Pages and ColdFusion
- Enterprise JavaBeans for bundling business logic in server-side components
- JDBC data access for scrollable database queries (result sets)
- JavaMail to send and receive mail with SMTP, POP3, or IMAP4 protocols
- JAXP for parsing XML documents
- Java Message Service for asynchronous communication between enterprise applications

Which Version Should You Use?

Well, of course, you want the latest and greatest version of Java! But, depending on where your Java programs are executing, choosing the latest Java version may not be possible in all situations. In short, consider the following when choosing a Java version:

- **Applets—** For applets, you'll want to go with Java Development Kit, Version 1.1 (JDK 1.1; last version is JDK 1.1.8_005). Netscape 4.06 and later and Internet Explorer 4.01 and later do not support Java versions later than 1.1 without a plug-in. Note that the latest release of Netscape, Version 6, is an exception to this rule and does support JDK 1.3. Regardless, for applets delivered over the Internet you shouldn't make assumptions about which browser the client is using, so you should write applets specific to the JDK 1.1 API.
- **Applications—** For stand-alone applications, you'll want to go with JDK 1.3, marketed as Java SDK, Standard Edition, Version 3.0. If you're writing server-side programs that rely on other vendor products, check which version of the JDK the vendor product supports.

Your best approach is to go with the JDK 1.3 but to bookmark the JDK 1.1 API when writing applets so that you can make sure that you are using methods and classes that are supported by the majority of client browsers.

Whichever Version You Use

Certainly, the Java 2 Platform provides a lot of capability. Our intent is to present material essential to developing distributed programs using Java.

Chapters 7 and 8 cover the basic syntax of Java. Then, we focus on Java applets (Chapter 9) and teach you how to use layout mangers (Chapter 12) to improve graphical user interfaces developed with the AWT (Chapter 13) or Swing components (Chapters 14 and 15).

Once we've covered the basics, we move to distributed programs by first examining sockets to open TCP connections to other computers on the Internet (Chapter 17). Distributed programs are often multithreaded for efficient handling of client requests, so integrated throughout this material are examples of multithreading programming. We cover multithread and synchronization of data (to eliminate race conditions that can occur between shared resources) in Chapter 16.

We next head over to server-side and enterprise programming, teaching you about Java servlets (Chapter 19) and JavaServer Pages (Chapter 20). This material covers cookies, sessions, and JavaBeans. Afterwards, we introduce you to database queries with JDBC (Chapter 22). Finally, we wrap up enterprise Java programming with XML and JAXP (Chapter 23) for processing platform-independent, business transactions (order requests).

After you've tackled this material, you'll have a firm background for developing distributed applications with Java and for diving into advanced topics like Enterprise JavaBeans, JNDI, and JavaMessaging.

6.4 Getting Started: Nuts and Bolts

Okay, okay, enough talk. Let's get on with it.

If you're wise, you won't sit down and read these Java chapters straight through, engrossing though they may be to you. `<SARCASTIC>`No doubt it will be difficult to tear yourself away, but you've got to do it.`</SARCASTIC>` Seriously though, we suggest installing Java as soon as possible, reading a little, practicing a little, reading a bit more, trying a more complex application, and so on. Write some real programs as *soon* as possible, and experiment with as many techniques as possible. There's no substitute for experience. Here's how to start:

- Install Java.
- Install a Java-enabled browser.
- Bookmark or install the on-line Java API.
- Optional: Get an integrated development environment.
- Create and run a Java program.

Install Java

Java is already bundled with some operating systems (e.g., OS/2, MacOS 10, Solaris 2.6), so you may have Java on your system already. If not, there are *free* versions of Java for Windows, MacOS, OS/2, Novell IntranetWare, Solaris, Irix, HP-UX, AIX, SCO Unixware, Linux, Amiga, BeOS, and most other major operating systems. Following is a list of a few of the most important download sites and versions available. For other operating systems and for-fee systems, check out Sun's list of Java ports at

http://java.sun.com/cgi-bin/java-ports.cgi

Note that each of these URLs, like every URL listed in the book, is available on-line at http://www.corewebprogramming.com/.

Java SDK, Standard Edition, Version 1.3 (JDK 1.3)

Microsoft Windows

http://java.sun.com/j2se/1.3/download-windows.html

Solaris SPARC/x86

http://java.sun.com/j2se/1.3/download-solaris.html

Linux x86

http://java.sun.com/j2se/1.3/download-linux.html

Java SDK, Standard Edition, Version 1.2 (JDK 1.2)

Microsoft Windows

http://java.sun.com/products/jdk/1.2/download-windows.html

Solaris SPARC/x86

http://java.sun.com/products/jdk/1.2/download-solaris.html

Linux x86

http://java.sun.com/products/jdk/1.2/download-linux.html

Java Development Kit, Version 1.1 (JDK 1.1)

Microsoft Windows

http://java.sun.com/products/jdk/1.1/download-windows.html

Solaris SPARC/x86

http://java.sun.com/products/jdk/1.2/download-jdk-solaris.html

Install a Java-Enabled Browser

This step will let you run Java programs embedded in Web pages (applets). Many IDEs and free versions of Java include "appletviewer," a mini-browser that ignores all of the HTML except for the applets. This is a quick way to test applets. For a fuller test, you'll want Netscape Navigator or Communicator, Microsoft Internet Explorer, Sun's HotJava, or another Java-enabled browser. This is a bit of a chicken-and-egg problem, since many of the download sites are accessible only by HTTP, which won't help you much if you don't have a browser already. For other platforms, presumably *some* browser came with your system or was provided by your ISP. If not, try using Netscape's anonymous FTP site.

Netscape Navigator

http://home.netscape.com/download/

Microsoft Internet Explorer

http://www.microsoft.com/ie/download/

Sun's HotJava

http://java.sun.com/products/hotjava/

Bookmark or Install the On-Line Java API

The official Application Programmer's Interface (API) describes *every* nonprivate variable and method in *every* standard library, something neither this nor any other single book can do. HTML versions for JDK 1.1, 1.2, and 1.3 are available at Sun and are bundled with many IDEs. The API can be accessed on-line directly from Sun's site, but the serious developer with plenty (5–10 MB) of extra disk space will want to install a local version for faster access.

Java 2 SDK, Version 1.3 (JDK 1.3)

API Specification

http://java.sun.com/j2se/1.3/docs/api/

API Download

http://java.sun.com/j2se/1.3/docs.html

Java 2 SDK, Version 1.2 (JDK 1.2)

API Specification

http://java.sun.com/products/jdk/1.2/docs/api/

API Download

http://java.sun.com/products/jdk/1.2/download-docs.html

Java 1.1(JDK 1.1)

API Specification

http://java.sun.com/products/jdk/1.1/docs/api/packages.html

API Download

http://java.sun.com/products/jdk/1.1/#docs

Note that you can find a list of all Java products available through Sun at http://java.sun.com/products/.

Optional: Get an Integrated Development Environment

In addition to a Java compiler and run-time system, you may want an integrated environment with a graphical debugger, class browser, drag-and-drop GUI builder, templates/wizards for database connectivity, and so on. A wide variety of IDEs are available on the market. You may want to look at John Zukowski's collection of IDE reviews and download sites at http://java.miningco.com/msub9.htm. A couple of popular IDEs are listed below.

Borland JBuilder

http://www.borland.com/jbuilder/

IBM VisualAge

http://www-4.ibm.com/software/ad/vajava/

Oracle JDeveloper

http://www.oracle.com/ip/develop/ids/jdeveloper.html

WebGain Visual Café

http://www.visualcafe.com/Products/VisualCafe_Overview.html

Sun Forte Developer

http://www.sun.com/forte/ffj/

Create and Run a Java Program

Create the File Write and save a file (say, `Test.java`) that defines the public class `Test`. Note that the filename and classname are case sensitive and must match exactly. If you are not using a Java development environment, use the text editor of your choice. Section 6.5 gives some simple examples.

Compile It If you are using the standard `javac` compiler from the Sun JDK on Windows or Unix, compile `Test.java` using `javac Test.java`. On a Mac, drag the source file onto the Java compiler. If you are using an Integrated Development Environment, refer to the vendor's instructions. The compilation creates a file called `Test.class`.

Run It For a stand-alone Java application with a command-line interface, run it by `java Test`. Note that this is `java`, not `javac` and that you refer to `Test`, not `Test.class`. On a Mac, drag the class file onto the Java runner. For an applet that will run in a browser, run it by loading the Web page that refers to it. For example, if you want the file `Test.html` to run the applet, then `Test.html` needs to refer to the URL of `Test.class` in an `<APPLET>` tag. We give details of this later.

6.5 Some Simple Java Programs

Following are some very basic programs to give a flavor of the language. Don't worry about understanding every detail; we'll go over things step by step later on. But it *is* a good idea to run these programs. Try making a few changes after successfully executing the original versions.

The Basic Hello World Application

"Application" is Java lingo for a stand-alone Java program. An application *must* contain a class whose name exactly matches the filename (including case) and that contains a `main` method declared `public static void` with an a single string array as an argument. A string array can be declared `String[] argName`, or `String argName[]`. Listing 6.1 presents a simple application that prints "Hello, world." when run. Additional application examples are given in Chapter 7 (Object-Oriented Programming in Java). Also, Java applications frequently use a graphical user interface. Section 9.10 (Graphical Applications) gives an overview, with more details given in Chapter 13 (AWT Components).

Core Approach

A public class named `SomeClass` must be defined in `SomeClass.java`. Case matters even on Windows 98/NT/2000; `SOMECLASS.java` or `someclass.java` will not work.

Listing 6.1 `HelloWorld.java`

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, world.");
  }
}
```

Compiling:

```
javac HelloWorld.java
```

Running:

```
java HelloWorld
```

Output:

```
Hello, world.
```

Command-Line Arguments

Listing 6.2 shows a program that reports on user input. This example looks a lot like C but illustrates a couple of important differences: `String` is a real type in Java, Java arrays have `length` associated with them, and the filename is not part of the command-line arguments. If you've never seen C or C++ before, you'll want to read the description of basic loops and conditionals given in Chapter 8 (Basic Java Syntax). Note that you *can* read command-line input on Macintosh systems even though there is no "command line"; in most implementations a small window pops up when the program starts to collect that input.

Listing 6.2 `ShowArgs.java`

```
public class ShowArgs {
  public static void main(String[] args) {
    for(int i=0; i<args.length; i++) {
      System.out.println("Arg " + i + " is " + args[i]);
    }
  }
}
```

Compiling:

```
javac ShowArgs.java
```

Running:

```
java ShowArgs fee fie foe fum
```

Output:

```
Arg 0 is fee
Arg 1 is fie
Arg 2 is foe
Arg 3 is fum
```

The Basic Hello World (Wide Web) Applet

"Applet" is Java lingo for a Java program that runs as part of a WWW page in a browser. Like an application, an applet must contain a class matching the filename, but applets don't use the `main` method. Instead, initialization is typically performed in the `init` method and drawing is done in `paint`. Listing 6.3 shows a simple Java applet that draws "Hello, World Wide Web." in a small window. Listing 6.4 shows the HTML document that loads it. Note that the name of the HTML file need not match the name of the Java file, but it is sometimes a useful convention. For more information on creating applets and drawing in windows, see Chapter 9 (Applets and Basic Graphics).

Listing 6.3 `HelloWWW.java`

```
import java.applet.Applet;
import java.awt.*;

public class HelloWWW extends Applet {
  private int fontSize = 40;

  public void init() {
    setBackground(Color.black);
    setForeground(Color.white);
    setFont(new Font("SansSerif", Font.BOLD, fontSize));
  }

  public void paint(Graphics g) {
    g.drawString("Hello, World Wide Web.", 5, fontSize+5);
  }
}
```

Listing 6.4 `HelloWWW.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>HelloWWW: Simple Applet Test.</TITLE>
</HEAD>
<BODY BGCOLOR="WHITE">

<H1>HelloWWW: Simple Applet Test.</H1>
<P>
<APPLET CODE="HelloWWW.class" WIDTH=460 HEIGHT=50>
  <B>Error! You must use a Java enabled browser.</B>
</APPLET>

</BODY>
</HTML>
```

Compiling:

```
javac HelloWWW.java
```

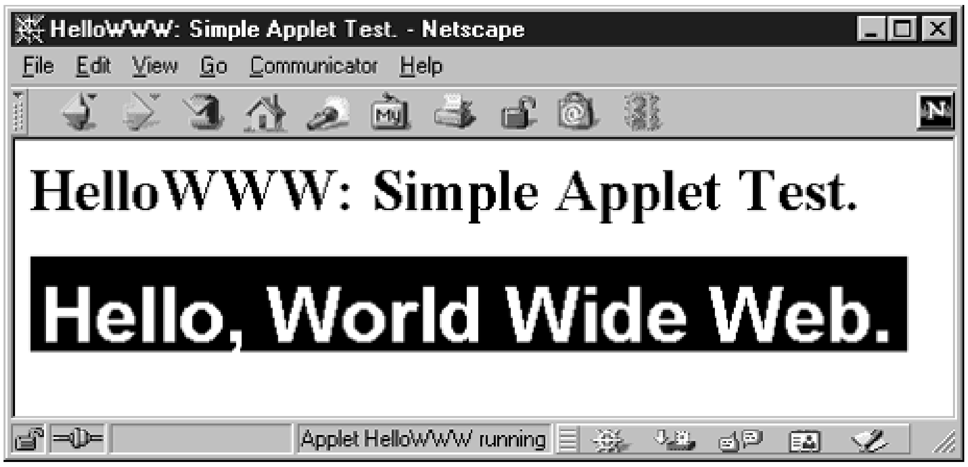
Running:

Load `HelloWWW.html` in a Java-enabled browser.

Output:

Figure 6-6 shows a typical result.

Figure 6-6. A simple applet, shown in Netscape Navigator 4.7 on Windows 98.


HelloWWW: Simple Applet Test. - Netscape
File Edit View Go Communicator Help
Applet HelloWWW running

HelloWWW: Simple Applet Test.

Hello, World Wide Web.

Applet Customization Parameters

Applets don't get command-line arguments because they are started by the browser. However, you can supply parameters to the applet by putting them inside `PARAM` elements between the `<APPLET ...>` and `</APPLET>` tags. The applet reads the values by calling `getParameter`. Listing 6.5 shows a variation of the `HelloWWW` applet that bases message text on `PARAM` values supplied. Listing 6.6 shows an HTML document that loads this applet four times with various messages. The use of `PARAM` is explained in detail in Section 9.7 (Reading Applet Parameters).

Listing 6.5 `Message.java`

```
import java.applet.Applet;
import java.awt.*;

public class Message extends Applet {
  private int fontSize;
  private String message;

  public void init() {
    setBackground(Color.black);
    setForeground(Color.white);

    // Base font size on window height.
    fontSize = getSize().height - 10;

    setFont(new Font("SansSerif", Font.BOLD, fontSize));
    // Read heading message from PARAM entry in HTML.
    message = getParameter("MESSAGE");
  }

  public void paint(Graphics g) {
    if (message != null) {
      g.drawString(message, 5, fontSize+5);
    }
  }
}
```

Listing 6.6 Message.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>The Message Applet</TITLE>
</HEAD>
<BODY BGCOLOR="WHITE">

<H1>The <CODE>Message</CODE> Applet</H1>
<P>
<APPLET CODE="Message.class" WIDTH=325 HEIGHT=25>
  <PARAM id="MESSAGE" VALUE="Tiny">
  <B>Sorry, these examples require Java</B>
</APPLET>
<P>
<APPLET CODE="Message.class" WIDTH=325 HEIGHT=50>
  <PARAM id="MESSAGE" VALUE="Small">
  <B>Sorry, these examples require Java</B>
</APPLET>
<P>
<APPLET CODE="Message.class" WIDTH=325 HEIGHT=75>
  <PARAM id="MESSAGE" VALUE="Medium">
  <B>Sorry, these examples require Java</B>
</APPLET>
<P>
<APPLET CODE="Message.class" WIDTH=325 HEIGHT=100>
  <PARAM id="MESSAGE" VALUE="Giant">
  <B>Sorry, these examples require Java</B>
</APPLET>

</BODY>
</HTML>
```

Compiling:

```
javac Message.java
```

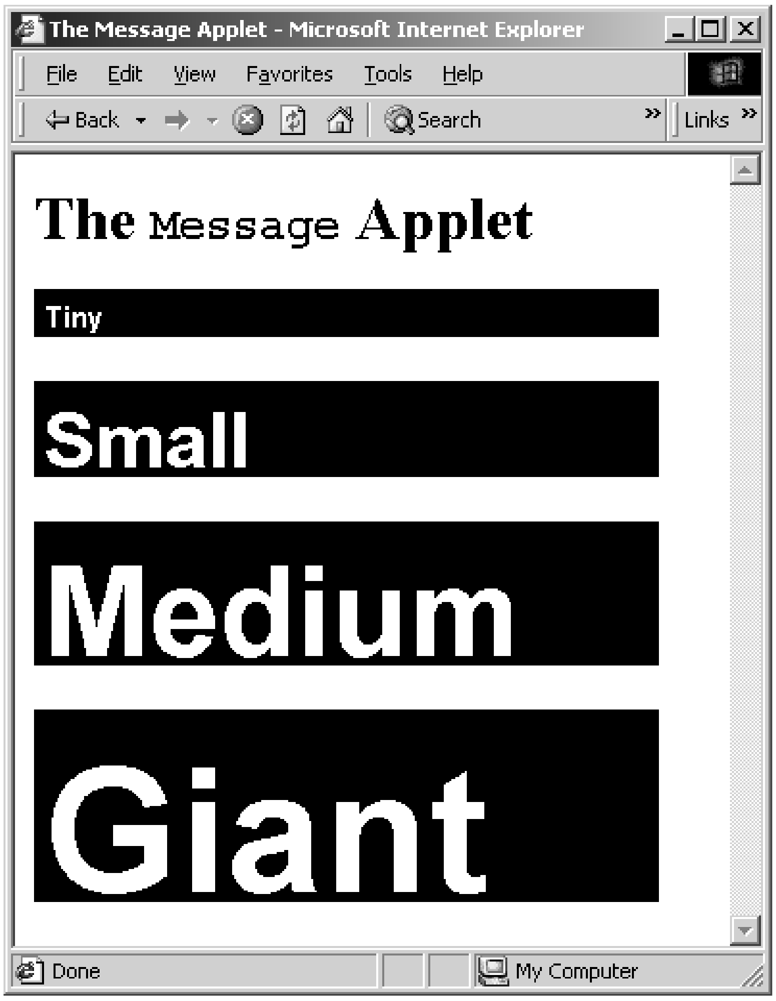
Running:

Load `Message.html` in a browser that supports Java.

Output:

Figure 6-7 shows the result of a Web page that loads the same applet four different times, supplying various `PARAM` values and differing `HEIGHT`s.

Figure 6-7. Four versions of the same applet, shown in Internet Explorer 5.5 on Windows 2000.


The Message Applet - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Links
The Message Applet
Tiny
Small
Medium
Giant
Done
My Computer

6.6 Summary

Java burst on the scene a few years ago and has been growing rapidly ever since. A number of features of Java will be new to most developers, even though many of the ideas were taken from existing languages, not invented just for Java. We hope this chapter helped you understand these and you now recognize the common misconceptions about Java as well. After installing Java, a Java-capable browser, and the Java API, you're ready to really get hacking.

Where you go next depends on your experience. If you've never used an object-oriented language before, you should carefully read Chapter 7 (Object-Oriented Programming in Java). If you're an OOP expert already, you can just skim the chapter to pick up on Java differences. Similarly, if you've never seen C or C++, you'll need to read through Chapter 8 (Basic Java Syntax) and try writing a number of programs. The experienced C/C++ hacker can quickly browse that chapter and move on. After that, the subsequent chapters cover graphics, windows, event-handling, threading, network programming, Java servlets, JavaServer Pages, and similar advanced Java topics.

CONTENTS