1 The History of Java

Way back in 1995, Sun Microsystems released the first version of the Java programming language to the public. Since then, Java technology has become an extremely popular language and has been adopted by millions of developers to create robust applications. Though Java applications can be executed in practically any environment, they are most commonly used in networked environments like an intranet or the Internet.

Java technology can be seen as both a language and a platform. The language allows applications to be written that can be run on practically any device, including a PC, a personal digital assistant (PDA), cellular phone, or television. The language is simple, secure, and powerful.

In this chapter you will learn

- Where Java technology came from
- The features of Java technology
- How Java compares to other languages
- How to download and install Java

Where Java Technology Came From

Back in 1991, some folks at Sun Microsystems were thinking about the future of computing. Their research indicated the "next big thing" would be intelligent consumer devices. A small group was formed under the codename "the Green Project" to create a prototype for a consumer device to try to get a jump on the market. The Green Project was essentially a secret project inside of Sun and they cut themselves off from the rest of the company to pursue their goals.

The Green Project

After 18 months of hard work, the Green Project team emerged with a device they called Star7. Essentially, it worked like a modern PDA (though it was a lot bigger) and had an animated, color touchscreen. It had an application running on it with a character named Duke that reacted to user prompts to do various tasks. The Star7 device was only meant as a demonstration platform, however. The Green Team expected that the software running on the Star7 could be deployed on dozens of different platforms like televisions, kiosks, and "smart" home appliances. In other words, the real power of the Star7 was the programming language that made it work.

One of the team members, James Gosling, created the language that made the demo work. He called this language Oak after a large oak tree outside his office window. The language was completely processor-independent so it could easily be used on all the different consumer devices that were available. One of the primary features of this software was that it could function nicely in a networked environment. The Green Project team tried to convince some new industries, including digital cable companies, that their creation was ideal. Unfortunately, at that time those industries were young and had murky visions of the future, so the entire Green Project was nearly dashed.

Enter the Web

However, in 1993 the World Wide Web absolutely exploded across the world, something that proved to be good fortune for the Green Project team and Sun. Since the language they created was designed to work over networks and provide dynamic content, the Internet suddenly seemed like the perfect environment in which the language could live on. They immediately realized they had created a programming language that had possibilities much larger than just

the consumer devices market. In fact, they had something that would change the way we used the Internet altogether.

In 1994, the Green Project began promoting their new programming language as a language-based operating system and targeted the online multimedia aspects of the Internet. They soon discovered that there was already an obscure programming language called Oak, so they had to change the name. Suggestions like Neon, Pepper, Silk, and Lyric were suggested before Java finally became the official name. Sun Microsystems began giving the language away on the Internet and finally made an official announcement of the language in May 1995.

Since that time, Java technology has become incredibly popular. We use it to provide dynamic content on the Internet, but it also has become a powerful language for developing large-scale enterprise applications and e-commerce applications. Java has even come full circle and is now embedded in many consumer devices like cellular phones, PDAs, and smart cards.

NOTE

Remember Duke, the character appearing on the original Star7 device? The Green Project team so loved the little guy that they kept him around and he has been the mascot for the Java programming language all along. Hey, it isn't every programming language that has a mascot!

The Features of Java Technology

Java has a long list of features that make it an excellent programming language. Java can be described as being simple, object oriented, interpreted, portable, robust, secure, multithreaded, and high performance. Beyond this, it also saves time and money and solves some important problems.

Those are all excellent traits for a language to have, so it is no surprise so many people are excited about Java and that it has become such a widely adopted programming language worldwide. The next few sections discuss each of these features of the language and show you why many people are so excited about programming with Java.

The Java platform is formed from two components: the Java Application Programming Interface (Java API) and the Java Virtual Machine (JVM). The Java API is a set of libraries that you can use to accomplish tasks like creating graphical user interfaces (GUIs), performing file input/output (I/O), or establishing network communication. The JVM is in charge of executing your code in a specific environment.

NOTE

The father of Java technology, James Gosling, along with Henry McGilton, wrote the official white paper on Java upon its original release in 1996. This paper discusses the features of the language in detail. You can check it out by going to `http://java.sun.com/docs/white/langenv/index.html`.

object-oriented
A programming methodology that organizes programs following a model of the real world. In the real world, objects are often composed of smaller components. In object-oriented development, this same concept is applied. This paradigm leads to flexible, reusable code.

object
A distinct unit of code in memory that, when combined with other objects, can form complete applications. Most programs will be composed of multiple objects that communicate with each other via methods.

state
The data of a program or application. An example might be the balance of a bank account. Subsequent functions would be used to operate on that state. For example, a method could calculate the interest on the balance passed to it.

behavior
The collection of methods for a particular class. The behavior of your objects typically manipulates their state.

Java Is Simple

Learning a new programming language is often a lot of hard work. I am not about to tell you that there will not be challenges to face in learning Java, but I can say that because of the stylistic simplicity of the language, I have seen many people learn it very quickly. Java does not have a lot of clutter to get in the way and, since it was built from the ground up with modern programming concepts in mind, there is a degree of familiarity for those who have worked with other languages.

One of the primary goals of the Java language was to remove any aspects of languages like C and C++ that were determined to be overly complicated and extraneous. Java does not have a large number of constructs, and this allows it to be very small and simple to understand. However, do not make the mistake of equating simplicity with inferiority; Java is a very powerful language indeed—much like English. After all, even though the English alphabet only has 26 letters, those letters can be combined in virtually limitless ways!

Part of ensuring Java's simplicity involved creating the syntax of the language itself. Java looks very much like C++, a language with which many developers are familiar. If you are a JavaScript programmer, when you go to write code, you will find working with Java very comfortable because the syntax is very similar. Don't think for a minute that this simplicity somehow results in a trivial language, though. Java is very powerful but its simplicity makes accessing this power very straightforward. After all, programmers want to produce solid code without having to twist their mind into knots trying to understand what they are doing. Java programs are easier to create than a comparable language like C++, but the results can be equivalently robust. You might even find yourself with a smile on your face once in a while when working with Java!

Java Is Object Oriented

The **object-oriented** paradigm has risen in popularity and has become the de facto standard for modern software development. An **object** is a model in software and contains qualities of both state and behavior. These objects can be used to represent anything you wish in a program.

For example, if you wanted to model an object like an airplane, you have qualities of **state**, like the number of seats in first class and coach, the amount of fuel the airplane holds, and the movie that will be shown. You would also have qualities of **behavior**, such as taking off, flying, turning, and landing, as part of your airplane object. Object-oriented programming focuses on state and behavior of individual objects. These objects can communicate with each other to form the complex logic necessary in most modern programs.

Earlier programming languages often used more **procedural code**, meaning the focus was on the behavior and not so much on the state. The concept was that the state only existed to support the behavior. So if you wanted to model the airplane with the procedural language, there would be **methods** that controlled the plane, making it take off, fly, turn, and land instead of the plane doing these things on its own. The object-oriented approach is much closer to how things work in the world.

If you have only worked with procedural programming prior to your venture into the world of Java, you will probably find learning the object-oriented concepts the most challenging aspect of your studies. Once your mind makes the paradigm shift to objects, though, you will see a whole new world of power and possibilities.

NOTE

If the concepts of objects and object-oriented programming are confusing to you, don't worry. You will learn more about this powerful methodology in Chapter 6, "Introduction to Object-Oriented Programming."

procedural code
This code is composed of a series of functions that perform distinct units of work on data passed to them. Procedural code is often difficult to manage and extend, though it is also generally easier to grasp initially than object-oriented code.

method
A unit of code that performs one or more actions. For example, an object may have a method named `print` that sends a document off to a printer somewhere. In other languages methods are sometimes called functions, procedures, and operations.

bytecode
The platform-independent format of compiled Java code that executes in the Java Virtual Machine.

Java Is Interpreted

Java source code is passed to a compiler that generates the **bytecode**. The bytecode is not targeted at any specific platform. Instead, a **Java Virtual Machine (JVM)** interprets the bytecode at runtime and executes it. This means that only the JVM itself is platform-dependent; the bytecode of your Java programs remains platform-independent.

This is different than a truly compiled language like C or C++. In a compiled language, platform-dependent information must be linked into the compiled code, forcing one compiled version for every target platform. If you write a program to calculate the distances of stars from each other and want it to run on Microsoft Windows, Linux, Sun Solaris, and Macintosh, you would have to compile it four times, once for each system.

The significant drawback with an interpreted language like Java is that code being dynamically interpreted executes more slowly than code that is compiled and native to a particular platform. Although this fundamental fact may be true, the JVM has been augmented over the years to become the **Java HotSpot Virtual Machine (Java HotSpot VM)**. The HotSpot VM contains an adaptive compiler that allows performance hot spots to be detected at runtime and optimized while your code is executing. This results in faster running code that still gains the benefits of being interpreted. Nowadays, properly designed Java programs execute at speeds comparable to similar programs written in C++. In essence, the one black mark of being interpreted has been removed completely.

Java Virtual Machine (JVM)
An abstract computing machine that all Java programs execute in. The JVM is the key to Java's cross-platform nature because it provides the same environment on any platform it actually runs on. The JVM is the intermediary between your Java code and the actual system on which the code executes.

Java HotSpot Virtual Machine (Java HotSpot VM)
The Java HotSpot Virtual Machine is specially tuned to provide optimum performance. It incorporates an adaptive compiler that allows code to be optimized as it executes. This means faster, more efficient code at runtime than past virtual machines have been able to achieve.

Java Is Portable

In the past, portability was not as much of a concern as it has become today. Most applications were fairly static in the sense that they were deployed on a consistent platform and did not require a lot of changes and tinkering to keep them running. However, in modern systems it is not at all uncommon for many components to be distributed across various hardware, operating systems, and networks. This heterogeneousness would pose great problems for many languages, but not Java!

Java applications can run practically anywhere. This makes Java quite revolutionary. Essentially anything that has some kind of processor can be Java-enabled, from mainframes to personal computers to telephones and beyond. Java programs are flexible enough to be local applications, web-based **applets**, server-side applications, and embedded software. The application code does not usually have to be changed to run on these different devices either. This means you can truly write the code once and run it anywhere you wish.

The key to this portability is the interpreted nature of the language. Since code does not have to be compiled to specific platforms, your Java programs can run anywhere a JVM exists. The world does not run on one type of platform alone and new platforms are constantly being introduced. By being portable, Java programs written today can still be viable tomorrow.

You can only achieve this portability for your Java code if you follow the rules, though. Believe me, it is entirely possible to write some very nonportable code if you are not careful. For this reason, Sun has introduced the 100% Pure Java initiative, which allows you to ensure that your code is portable by running it through a variety of test suites. Obviously, maintaining portability is considered extremely valuable; Java was designed with this important consideration in mind from the beginning. It is nice to know the engineers at Sun are always working to make your life a bit easier!

applets
Executable modules that are automatically downloaded to a user's web browser over a network like the Internet. Applets allow deployment to be simple and provide a mechanism to add advanced functionality to web pages.

Java Is Robust

Robust code is reliable code. Java has a few features that tend to make it more robust than many other languages, thus easing the burden on developers attempting to avoid pitfalls. Specifically, Java is strongly typed, includes automatic memory management, utilizes garbage collection, and provides an exception handling mechanism. Let's take a look at what these qualities mean.

Java is considered a **strongly typed** language. This means that the code has many checks made against it to ensure that it correctly follows the rules of the language. Java is very strict about what is considered legal in code, and the compiler simply does not allow you to make many of the mistakes that have plagued developers working in other languages for years. Essentially, the compiler enforces the rule that all declarations in your code are explicitly given. You cannot use a variable called *fred* without declaring what type *fred* actually is (perhaps an integer, or a byte, or an image). Similarly, you cannot invoke a method unless that method has already been defined, and you cannot pass any parameters to that method unless those have been explicitly listed.

The compiler is only half of the solution, however. The JVM also has a part to play in ensuring robustness. All memory is managed for you automatically, and it is impossible for Java code to stomp on your system memory, potentially corrupting parts of your data. The interpreted nature of the language allows the JVM to take full control of memory management, freeing developers from having to handle these complex details on their own. Java developers do not work with **pointers** at all, thus immediately removing the possibility of many complex bugs.

The runtime also includes the **garbage collection** mechanism. This is a part of the JVM that monitors memory and determines if there is any "garbage" to clean up. With C++, for example, you would have to ensure that all objects you used were removed from memory or risk the nasty business of a memory leak. With Java, all of this potentially complex memory cleanup and maintenance is handled for you automatically.

Java also includes an extensible **exception handling** mechanism, similar to the system used in C++. Instead of creating simple error codes and passing these around your programs, Java allows you to define exception types that signify specific error conditions. For example, you might wish to signify that a network port is unavailable or that there are not enough items in an inventory to fulfill an order. These possible error conditions can then be handled within the programs that are attempting to perform these actions. These errors could even be overcome without any user intervention whatsoever; the code can correct these errors while it is running. In other words, the exception handling aspect of the language allows you to maintain robustness in your specific programs as well.

strongly typed

When a language is strongly typed, it means that it imposes strict rules on the declarations made in the code itself. Some languages allow a variable to represent an unknown data type, but languages like Java force you to declare all variables to be a specific type before they can be used.

pointers

In languages like C and C++, a pointer represents a specific location in memory that is controlled by the code itself. Pointers can lead to dangerous problems, including data corruption, if they are not used correctly. Java removes the whole notion of managing your own pointers, which removes this often unnecessary complexity.

Java Is Secure

One of the top features for end users of a Java application is that it can be dynamically downloaded from a remote location. Although this is indeed a powerful, desirable feature, there is a lot of risk inherent in the process. It does not seem wise to download code from someone on the Internet and then just let it run at will on your system, does it? This is how things like viruses and other maliciousness can invade your otherwise happy computer. Luckily, Java recognized these potential threats and incorporated a multiphase approach to ensure a high level of security.

garbage collection
Part of the Java Virtual Machine's responsibility is managing memory on your behalf. When memory space you have used in your code is no longer needed, the garbage collection mechanism kicks in and eventually clears that memory automatically. Because of this automatic procedure, there is not a standard way to manually clear memory from within Java programs. The garbage collection process is a great benefit because it reduces both the amount of code you need to create and, more importantly, dangerous bugs that can creep into your code if you could otherwise mismanage memory.

To begin with, you just learned about the robustness of the language in regard to its memory management and compile-time checking. These two features also contribute significantly to the Java's security. Since the JVM is "in charge," it is impossible for normal Java code to cause a problem with system memory that could lead to insecure or corrupted data. It also means that pure Java code is unable to install a virus or worm on your system because it cannot "touch" memory directly.

Java security also extends to so-called "foreign" code. This does not mean code written in another country of course, but code executing across a network. If you access code over a network (as would be the case with a Java applet), the natural reaction of Java is to not trust the code whatsoever. This does not mean the code cannot execute, but it does mean that the code will not be able to access your local filesystem or devices like printers and modems. The end user can override this security, but the default behavior is to protect their systems from potential harm. This is the type of security most users are immediately concerned with because it prevents an invasion of their privacy.

Because Java technology is so prevalent on networks and in modern enterprise systems, its security aspects are absolutely vital. Because Java provides such excellent security inherently, it is the perfect language to be used in these environments and can be trusted not to cause more harm than good.

WARNING

Security is never perfect. It is unwise to ever consider anything on a network completely secure. This means that if hackers have enough motivation, they can defeat *any* security. The true goal of any security scheme is to make it so difficult to breech that it becomes practically pointless to continue trying. In all my years of working with security, I have found the best way to thwart attacks is to force the miscreants to just give up and go away. The Java language has established itself as being extremely secure, but this does not mean that just because you use the language your programs are totally protected from attack.

Java Is Multithreaded

If a program is **multithreaded**, it means it can do more than one thing at a time. Most applications you use are multithreaded, like your web browser, for instance. Imagine visiting a website that allows you to play a music clip while you are scrolling through information on a page. Since you are probably working on a machine with a single processor, only one thing can be done at a time. However, when an application is multithreaded, it means "lightweight processes" can execute concurrently. These lightweight processes are still only executing one at a time, but they are swapped out so quickly that it appears as if several things are happening instantly.

These lightweight processes are called **threads**, and each thread can be assigned a specific operation to perform for the master application. Since the Java language is natively a multithreaded language, utilizing threads in your programs is a straightforward task.

NOTE

Threads are an advanced concept and are not discussed further in this book. If you are interested in learning more about threads and multithreading, I recommend *Java Threads* by Scott Oaks and Henry Wong.

Java Is High Performance

In the past, Java had some performance issues that generated concerns for its viability, but recent versions have performed at the speed that developers demand. The perception that Java programs were slow was mainly because it is an interpreted language. An interpreted language has to read every individual instruction and compile it into instructions that are understood by the system on which the code is executing. These added steps have a tendency to slow down programs considerably. With all of the other great features the Java language provided, it was unfortunate, but not surprising, that speed was the major drawback. Speed is, after all, usually paramount on anyone's list of importance in an application.

However, advancements in the way the JVM works have made Java comparable to the speed of C++ code in most situations. This increase in horsepower was achieved without sacrificing any of the other important features, too! These improvements have made Java the perfect choice for the modern developer, especially when they are developing for the Internet and other networked environments.

exception handling
A form of flow control that is used to handle program errors. In many other languages, errors are reported as a code number of some kind that is often cryptic and difficult to work with. Java makes use of exception handling, which provides a more robust method for trapping and recovering from logical errors.

multithreaded
An application that can control individual threads to perform specific actions is considered multithreaded. By divvying up the processing across these threads, an application can appear to be performing multiple actions simultaneously. Java is inherently multithreaded, making the creation of these advanced programs simpler than other languages.

threads
Lightweight processes contained within an actual process. Threads are the building blocks of multithreaded programs and provide separate distinct processing.

An important point to keep in mind about Java's performance is that it performs wonderfully in most applications, including GUI-based solutions and network code. Those types of applications are not constantly executing but spend most of their time waiting for input, processing data, and returning output. Although Java is not a language designed to write low-level device drivers and the like, it is an excellent language for almost every other type of application.

NOTE

You may wonder why Java is not a good choice for writing device drivers. A device driver should be targeted to specific hardware to allow it to take full advantage of speed and free access to video cards, printer ports, etc. Since Java is a cross-platform language, it is best suited for higher-level applications that do not directly access hardware-specific features. So if you were hoping to write a Java video driver, you may want to rethink your plans a bit!

Java Saves Time and Money

By using Java technology, you can often significantly reduce the cost of development. Software development with Java is a much quicker process than with most other languages. This is largely because of the platform independence and familiarity of the language. Because the syntax is very similar to popular languages like C/C++ and JavaScript, it is usually relatively simple for developers to learn how to work with Java technology, and because of the widespread industry support, it is often convenient to incorporate Java applications into an existing infrastructure.

Deployment costs are also lowered and often eliminated altogether. It is very common for Java applications to be delivered dynamically over a network without any user intervention whatsoever. This means the end user does not have to install, configure, or otherwise tinker with the application they wish to use.

Java Solves Important Problems

Java was designed for networking and security from the very beginning. When applications are used over a public, insecure network like the Internet, it poses problems for these dynamic applications. Who would use online banking software that was not secure? What user wants to install dozens of different applications on their system to access online applications? From the business perspective, how can you handle problems like deployment and version control and also be sure that the code cannot be used in a dangerous fashion to exploit your sensitive corporate information?

Java is perfect for solving many of these modern problems. Many businesses use the public Internet backbone to deploy websites for their prospective consumers; often these businesses want to provide enhanced functionality. This might include account management at a bank, or three-dimensional modeling of automobiles, or even a virtual desktop that allows employees to work from remote locations. All of these types of applications—and many more—can be created with Java technology.

How Java Compares to Other Languages

Though Java is a great language, it is one of the newer kids on the block. Other languages like C++, JavaScript, and Perl are also popular choices made by developers. Java has commonality with each of these as well as many differences. In this section, you see how Java stacks up head-to-head with these languages and why Java might be a better choice.

We can categorize programming languages into three categories. First are the compiled languages like C and C++ that are bound to each native platform, making them high performance but not very portable. Second are the scripting languages like JavaScript and Python that are useful for portable processing but have little to offer in the way of performance. Third, and somewhere in between those first two categories, is the Java language itself. It combines the portability and simplicity of the scripting languages, yet it can execute at speeds comparable to the compiled languages. This unique combination of traits makes Java able to solve many problems in many different situations.

Java and C++ **C++** is another popular, object-oriented language that can be used to create high performance code. In fact, most enterprise development that is not being written in Java is probably being written in C++. As a language, C++ is syntactically fairly simple, though it is not as user-friendly as Java tends to be. C++ is also truly compiled code, so it cannot just be run on any platform you wish without specifically building a binary executable for your system.

Although both languages are classified as object oriented, Java is the one that was designed with those concepts in mind from the beginning. This contributes to Java's flexibility, syntactic simplicity, and overall "friendliness."

Java and JavaScript It has been said that the only thing the Java and **JavaScript** languages really have in common are their first four letters. That is not exactly true, but it does indicate that the similarity is not as strong as you may have thought. While Sun Microsystems created the Java programming language, JavaScript was created by Netscape. JavaScript is a scripting language

C++
An object-oriented version of the C programming language that gained immense popularity in the early 1990s. C++ can be thought of as a very close cousin to the Java programming language.

JavaScript
A scripting language developed by Netscape to add interactivity to web documents. JavaScript is a programming language but is very simple to learn and use, making it excellent for web content developers who may not have backgrounds with more complex programming languages.

that is based on object-oriented concepts, but it does not treat objects exactly the same way as a language like Java does.

Java and Perl Perl is another scripting language used in a wide variety of scenarios. Perl really became popular when it became known for extending websites and allowing dynamic processing. Perl is a very common implementation language for **Common Gateway Interface (CGI)** solutions and is prevalent on the Internet. Perl is one of those languages you either love or hate. It is very flexible and powerful, but it is also perceived as somewhat complicated. Java can do anything Perl can do functionally, but it was designed to be more user-friendly than Perl seems to those new to the language.

Common Gateway Interface (CGI)
A standard for interfacing external applications with HTTP servers on the World Wide Web. CGI solutions are often used to provide functionality to a website like form processing, image creation, and dynamic HTML generation.

How to Download and Install Java

Before you actually download and install Java on your system, let's take a look at some of the various available downloads.

The Java 2 Platform Standard Edition (J2SE) The J2SE is the essential download you need to both develop and execute Java programs. It contains the API libraries, the compiler used to produce bytecode, the JVM and interpreter, and various other tools.

The Java 2 Runtime Environment, Standard Edition (J2RE) The J2RE is a trimmed down version of the J2SE, basically. It does not contain the compiler or other tools, but it still has the API libraries and the interpreter. This download should be installed on a system that needs to execute Java code only. It does not provide the mechanisms needed to create Java applications, only to run them.

NOTE

There is something called the Java Plug-in that provides the runtime environment for most users of online Java applications. Essentially, the Java Plug-in is an automatically deployed version of the J2RE. If you download the J2SE, the Java Plug-in is actually included right along with everything else. You can learn more about the Java Plug-in by pointing your browser to `http://java.sun.com/products/plugin`.

The Java 2 Platform, Enterprise Edition (J2EE) J2EE is both a platform and a technology, designed to make the creation of highly scalable enterprise applications simpler. J2EE relies on various vendor-provided servers to provide the execution environment. Note that J2EE still requires the standard APIs found in the J2SE to operate.

JavaServer Web Development Kit (JSWDK) The JSWDK is used for writing and testing servlets or JavaServer Pages (JSP), which let you extend a web server more powerfully than traditional CGI solutions.

The Java 2 Platform, Micro Edition (J2ME) The J2ME is an optimized and smaller version of the J2SE that is designed to produce programs capable of running on consumer devices like smart cards, cell phones, and PDAs. It supports networked as well as standalone applications, user interfaces, and security.

There are some other Java technology-related downloads, but these are the five major bundles available. The remainder of this book deals only with the first download, the J2SE itself. The good news is that every one of the other Java downloads is built on the fundamentals you will learn in this book, so you will eventually be able to investigate all of them at some point.

Downloading the J2SE Software

So let's get started! Be sure you are connected to the Internet. Then open your browser and enter the URL `http://java.sun.com/j2se/1.4.1/download.html` and you will find a list of downloads for Windows, Linux, and Solaris as shown below.

Download J2SE™ v 1.4.1_01	JRE	SDK
Windows (U.S. English only)	DOWNLOAD	N/A
Windows (all languages, including English)	DOWNLOAD	DOWNLOAD
Linux RPM in self-extracting file	DOWNLOAD	DOWNLOAD
Linux self-extracting file	DOWNLOAD	DOWNLOAD
Solaris™ SPARC™ 32-bit self-extracting file	DOWNLOAD	DOWNLOAD
Solaris SPARC 32-bit packages - tar.Z	N/A	DOWNLOAD
Solaris SPARC 64-bit self-extracting file *	DOWNLOAD	DOWNLOAD
Solaris SPARC 64-bit packages - tar.Z *	N/A	DOWNLOAD
Solaris x86 self-extracting file (Solaris 7, Solaris 8)	DOWNLOAD	DOWNLOAD
Solaris x86 packages - tar.Z (Solaris 7, Solaris 8)	N/A	DOWNLOAD
Installation Instructions	VIEW	VIEW
ReadMe	VIEW	VIEW
Release Notes	VIEW	VIEW
License for all platforms	VIEW	VIEW

As you can see, there are multiple download options for each platform, including the choice between the Java Runtime Environment (JRE) and the Java 2 Software Development Kit, Standard Edition (J2SDK). We are specifically interested in the SDK choices.

When you download the SDK for the platform you are interested in, you are taken to a license agreement. After reading it, select the Accept button and you are taken to a page with a link to the download itself. Just click and wait until the download completes.

WARNING

These downloads are rather large, so be sure your Internet connection is available for the time required.

Once you have the download on your system, you are ready to install. I have provided directions for the Windows and Linux platforms in the following sections.

NOTE

Instructions for installing J2SE on Solaris can be found at `http://java.sun.com/j2se/1.4.1/install-solaris.html`.

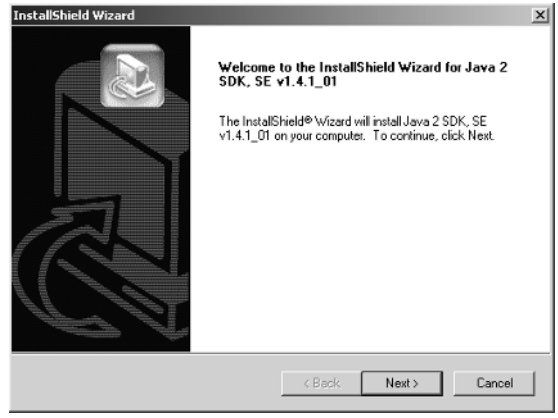
Installing J2SE on Microsoft Windows

The J2SE is supported on Windows 98, Windows ME, Windows XP, Windows 2000 Professional, and Windows NT 4.0. You should ensure that there are at least 120 megabytes of space available in your filesystem before you begin the following steps.

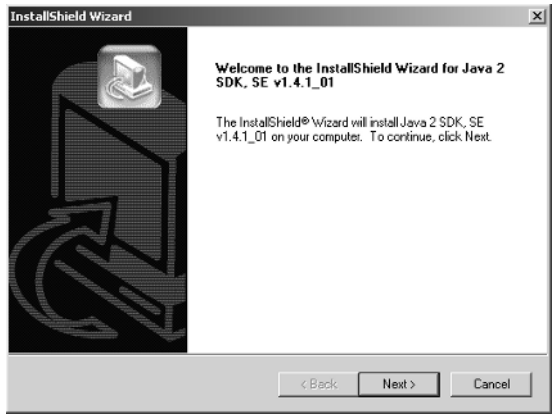
NOTE

These directions are specific to Microsoft Windows XP. The installation procedure is effectively identical on all the supported versions of Windows, but there might be small differences in the appearance of some of the screens.

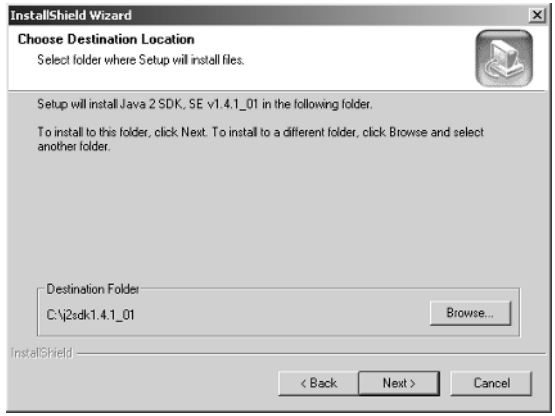
1. After the executable extracts the required contents to a temporary directory and the InstallShield wizard welcome screen appears, click the button labeled Next.



2. The license agreement appears. When you are done reading through this, click Yes.

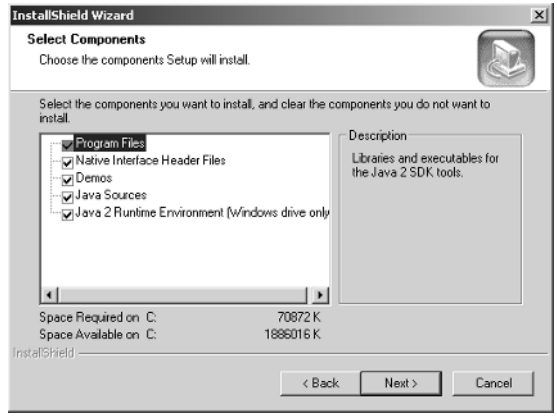


3. The next screen allows you to select the directory where you want the J2SE to be installed. If you want to choose a location other than the default one presented here, click the Browse button and choose or create a directory. Once you have decided on the installation directory, click Next.

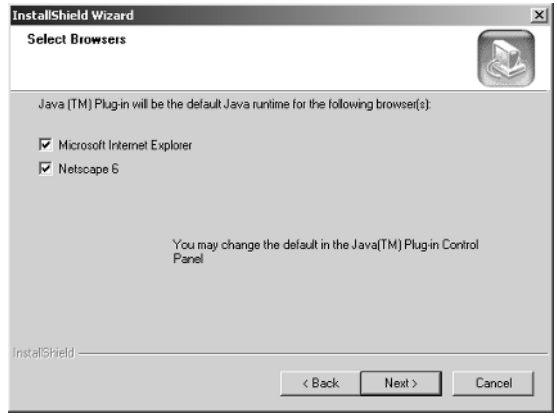


4. The next screen shows a list of components you can install. The Program Files option is mandatory, but the remaining choices are optional. The Native Interface Header Files refers to files needed to work with C code. If you have the Demos option selected, a large assortment of code is installed under the destination directory that lets you see many samples of complete Java code. The Java Sources option installs the actual Java source files for every class in the Java APIs. The final option, Java 2 Runtime Environment, installs the Java Plug-in if selected.

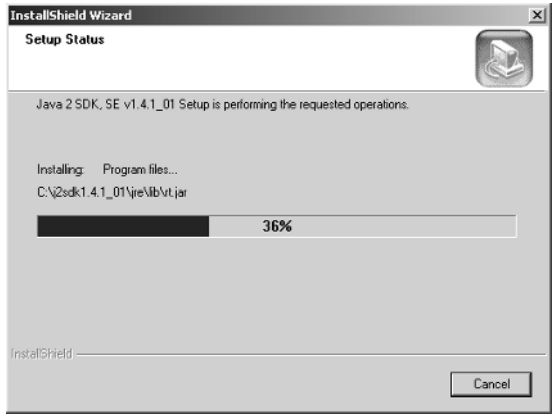
It is usually appropriate to just leave all of the options selected, but you may want to ensure that you have enough space available. Just under this list you can see the space required and the space available on your hard drive. You can deselect some of the optional installation items if you want to reduce the size of the installation. Once you are all set, click Next.



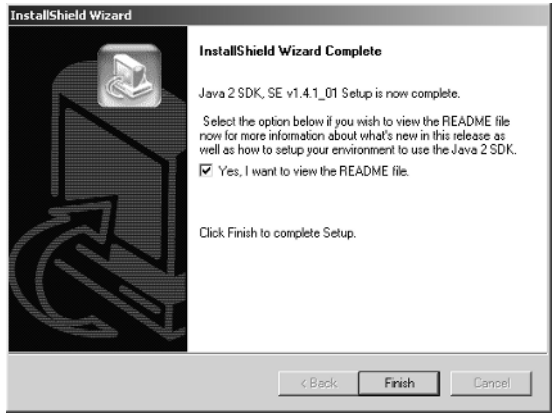
5. If you selected the Java 2 Runtime Environment option in step 4, you see the browser selection screen. You can deselect browsers on this screen if you do not want the Java Plug-in to be automatically associated with them. Click Next.



6. The installation now begins. As is typical, you see the progress bar as the files are installed on your system. You may also see the Java 2 Runtime Environment setting up automatically (assuming you selected this option in step 4).



7. Once all the files have been installed, you see the final screen indicating success. If your system needs to be rebooted to complete the installation, this screen informs you of that fact. Click the button to complete the installation.



You have now installed the J2SE. Now you have to configure Java for your environment.

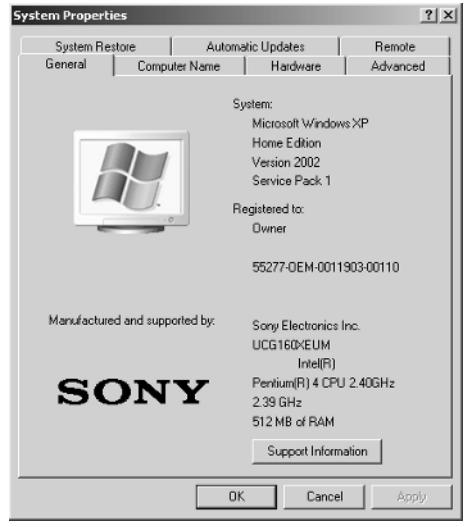
Configuring the Installation on Windows

Java should now be installed on your system, but there are still a few things left to take care of. To be able to use the Java compiler and runtime, you first need to update your PATH environment variable. You can accomplish this in different ways, depending on your specific version of the Windows operating system.

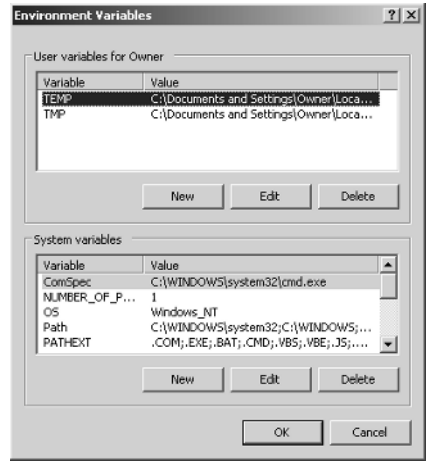
NOTE

These configuration steps are specific to Windows XP. If you are using a different version of Microsoft Windows, you can find specific configuration instructions at `http://java.sun.com/j2se/1.4.1/install-windows.html`.

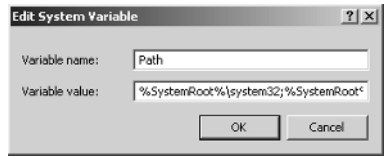
1. Right-click the My Computer icon and select Properties from the pop-up menu. This opens the System Properties window.



2. Select the Advanced tab and click the button labeled Environment Variables. This opens the Environment Variables window.



3. In the System Variables section located in the bottom half of this window, select the PATH environment variable and click the button labeled Edit. This opens the Edit System Variable dialog box.

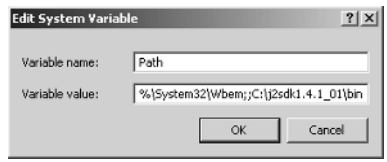


4. Append the following line to your Path statement:

```
C:\j2sdk1.4.1_01\bin
```

WARNING

This may not be the exact path to your installation directory. You should replace the installation directory with whatever selection you made back in step 3 of the "Installing J2SE on Microsoft Windows" section. Be sure to include the *bin* subdirectory in your path, though.



5. Click the OK button to clear the Edit System Variable dialog, then click the OK button to clear the Environment Variables window, and finally click the OK button to close the System Properties window.

Congratulations! You have installed Java on your system and the configuration is now complete.

Installing J2SE on Linux

If you are running the Linux operating system, you do not have the advantage of a wizard-based installation. Even so, the procedure is not difficult and this section should help you get Java installed and configured for your system.

NOTE

These instructions are for the automatic installation provided via the Linux RPM file. This installs Java to the `/usr/java` directory by default and requires you to become root to complete the installation. If you need to have more control over which directory Java installs to or you do not have root access, follow the instructions for installing a self-extracting binary at `http://java.sun.com/j2se/1.4.1/install-linux.html`.

1. Download the Linux RPM in the self-extracting file from `http://java.sun.com/j2se/1.4.1/download.html`. Be sure to grab the SDK, not the JRE, since you want the compiler to be installed as well. The filename should be similar to `j2sdk-1_4_1_01-linux-i586-rpm.bin`.

WARNING

This download is rather large, so be sure your Internet connection is available for the time required.

2. Open a terminal window and change to the directory where the RPM download is located.
3. You need to make the file executable, so type this command at the prompt in your terminal window:

   ```
   chmod a+x j2sdk-1_4_1_01-linux-i586-rpm.bin
   ```

4. To unpack the downloaded RPM, type the following in the terminal window:

   ```
   ./j2sdk-1_4_1_01-linux-i586-rpm.bin
   ```

   This displays a license agreement. Once you have read the agreement, agree with the terms and the RPM is extracted to the current directory.
5. You need to become the root user to continue, so switch to **`su`** now and enter your root password when prompted.
6. To install the download automatically, type this command in the terminal window:

   ```
   rpm -iv j2sdk-1_4_1_01-linux-i586-rpm.bin
   ```

7. You can verify that the installation was successful by checking the version of the Java runtime. Type the following line in the terminal window:

   ```
   /usr/java/j2sdk1.4.1_01/bin/java -version
   ```

   This should return the version number of the download you installed (in this case, the version will begin with 1.4.1). Once you have successfully installed Java, you can move on to the next section to configure your system.

Configuring the Installation on Linux

Now that Java in installed on your Linux system, you need to set your PATH environment variable to include the directory where the Java tools are stored.

1. Open your shell's startup script (for example, the `.cshrc` or `.kshrc` file).
2. Set up the JAVA_HOME environment variable. How this is done will depend on which Linux shell you are using, but the value of JAVA_HOME should be set similar to the following:

   ```
   /usr/java/j2sdk1.4.1_01
   ```

WARNING

The value is step 2 may be different if you have installed the SDK to a different directory or if you are using a different SDK version. If you are not sure what the exact value should be, consult the online installation instructions.

3. Now add the PATH environment variable. Note that you may already have a PATH set in your startup script. This command adds the required path to the Java tools to your existing PATH environment variable. You may want to check the installation instructions for setting this value, but the value you need to add will be similar to the following:

   ```
   $JAVA_HOME/bin:$PATH
   ```

4. Close your startup script and restart your system. Once the system is rebooted, your Java installation should be configured and ready to go.

TIP

If you are familiar with your Linux shell, you do not have to reboot the entire system. Instead you can run your startup script using the specific "source" command for your system. Consult your system documentation for more information on refreshing your startup script.

Review Questions

Terms to Know

- ❑ applets
- ❑ behavior
- ❑ bytecode
- ❑ C++
- ❑ Common Gateway Interface (CGI)
- ❑ exception handling
- ❑ garbage collection
- ❑ Java HotSpot Virtual Machine (Java HotSpot VM)
- ❑ Java Virtual Machine (JVM)
- ❑ JavaScript
- ❑ method
- ❑ multithreaded
- ❑ object
- ❑ object-oriented
- ❑ pointers
- ❑ procedural code
- ❑ state
- ❑ strongly typed
- ❑ threads

1. What are the two components that form the Java platform?

2. What was the name of the internal project at Sun Microsystems that produced the first version of the Java programming language?

3. What are some types of applications for which Java is suited?

4. Who is considered the father of Java technology?

5. What does it mean for a language like Java to be strongly typed?

6. What does the Java compiler produce from source code?

7. What is the engine that allows Java code to be platform-independent?

8. To what modern development paradigm does the Java language adhere?

9. What is included in the Java 2 SDK Standard Edition?

10. What language was Java syntax largely based on?