The Java™ Platform

A White Paper

Douglas Kramer

With contributions by
Bill Joy and David Spenhoff

JavaSoft
A Sun Microsystems, Inc. Business

2550 Garcia Avenue
Mountain View, CA 94043 U.S.A.
408-343-1400

May 1996

The Java™ Platform

The Java™ Platform

This paper defines the Java™ Platform and provides descriptions of each of its parts. It is written for developers and others interested in understanding the wide range of environments where Java-powered applets and applications can run.

For more up-to-date and detailed information about the features and architecture of the Java Platform and development environment, refer to the Developer's Corner at the JavaSoft web site `http://java.sun.com`. In particular, see *The Java Language Environment: A White Paper.*

The Java Platform is developed by JavaSoft, an operating company of Sun Microsystems, Inc.

What is the Java Platform?

The computer world currently has many platforms, among them Microsoft Windows, Macintosh, OS/2, UNIX® and NetWare®; software must be compiled separately to run on each platform. The binary file for an application that runs on one platform cannot run on another platform, because the binary file is machine-specific.

The Java Platform is a new software platform for delivering and running highly interactive, dynamic, and secure applets and applications on networked computer systems. But what sets the Java Platform apart is that it sits on top of these other platforms, and compiles to *bytecodes*, which are not specific to any physical machine, but are machine instructions for a *virtual machine*. A program written in the Java Language compiles to a bytecode file that can run wherever the Java Platform is present, on *any* underlying operating system. In other words, the same exact file can run on any operating system that is running the Java Platform. This portability is possible because at the core of the Java Platform is the Java Virtual Machine.

While each underlying platform has its own implementation of the Java Virtual Machine, there is only one virtual machine specification. Because of this, the Java Platform can provide a standard, uniform programming interface to applets and applications on any hardware. The Java Platform is therefore ideal for the Internet, where one program should be capable of running on any computer in the world. The Java Platform is designed to provide this "Write Once, Run Anywhere"℠ capability.

Developers use the Java Language to write source code for Java-powered applications. They compile once to the Java Platform, rather than to the underlying system. Java Language source code compiles to an intermediate, portable form of bytecodes that will run anywhere the Java Platform is present.

Developers can write object-oriented, multithreaded, dynamically linked applications using the Java Language. The platform has built-in security, exception handling, and automatic garbage collection. Just-in-time compilers are available to speed up execution by converting Java bytecodes into machine language. From within the Java Language, developers can also write and call native methods—methods in C, C++ or another language, compiled to a specific underlying operating system—for speed or special functionality.

The Java Language is the the entry ramp to the Java Platform. Programs written in the Java Language and then compiled will run on the Java Plaform. The Java Platform has two basic parts:

- Java Virtual Machine
- Java Application Programming Interface (Java API)

These are described in detail later in this paper. Combined, these parts provide an end-user runtime environment for deploying Internet and intranet applications.

The Java Base Platform

The Java Base Platform is the *minimum* Java Platform that developers can safely assume is present for running Java-powered applets and applications. This platform applies to Network Computers, desktop computers, and workstations (the next section describes the platform for smaller systems). This platform contains the same Java Virtual Machine mentioned before, but has a minimal set of API required to run basic applets and applications. This minimal set is known as the Java Applet API, or Java Base API. Developers who write to this minimum set can feel secure that the program will run anywhere without the need for additional class libraries.

Certain Java Platform licensees (listed in "Where Will the Java Platform Be Deployed?" on page 10") have contracted to include the Java Base API in their particular implementation of the Java Platform. As more class libraries are developed, the Java Base Platform will grow, and these additions will migrate in a timely fashion into the Java Base Platform present on each licensee's operating system.

Another set of APIs, called the Standard Extension API, is being defined by JavaSoft, in partnership with leading industry companies, to extend the base functionality. Over time, some subset of the Standard Extension API will migrate into the Java Base Platform.

The Embedded Java Platform

The Embedded Java Platform is being targeted for consumer devices with fewer resources and more specialized functionality than a Network Computer, such as set-top boxes, printers, copiers, and cellular phones. Such devices might have special constraints such as small memory footprint, no display, or no connection to a network.

The API targeted for this platform is called the Java Embedded API. The Java Embedded API is the smallest API a low-function embedded device can have and still run. Because this platform is still under development, this API has not yet achieved the level of a standard. Consequently this API is not yet well-defined, but it will probably consist of the packages `java.lang` and `java.util`. A Java-powered application written for one particular device could operate on a wide range of similar, dedicated devices.

Benefits of the Java Platform

The Java Platform has benefits for the end-user as well as the developer and support personnel:

End-User Benefits

Today, the Java Platform provides live, interactive content on the World Wide Web, with just-in-time software access. Applications are readily available on all operating systems at once, freeing users from having to choose operating systems on that basis. Smaller, less expensive, dedicated systems will eventually be available for specialized applications.

Developer Benefits

The Java Language is a small, "knowable" system and is coupled with a growingly comprehensive set of APIs. Developers can "Write Once, Run Anywhere," which provides tremendous marketing leverage over other languages. In addition, Java development environments on all operating systems compile to a single binary format. Rather than developing on multiple platforms to deliver on multiple platforms, developers can now develop on one platform, saving cost, to deliver on that same platform, which is everywhere. The ability to "Write Once, Run Anywhere" is enough reason for some developers to turn to the Java Language as an alternative to C or C++ even for stand-alone, non-networked applications.

In addition, building applications from shared, reusable objects can further reduce cost by allowing developers to concentrate on creating only what is novel. Developers can distribute by network rather than compete for shelf-space in software stores.

Administrative and Support Benefits

The Java Platform has benefits for corporate computer systems administration departments. Version control and upgrades are simplified because Java-powered applications can be kept in a central repository and served from there for each individual use. In multivendor, multiplatform environments, the number of platforms to support is reduced to one. Emerging lower-cost network computers have the potential to reduce maintenance and capital expenditures. With these network computers, data management can remain centralized while data processing is done locally.

Companies with large intranets, that may not find it worthwhile to upgrade to the latest memory-consuming operating system, can run Java-powered applications on all their existing machines. By providing corporate data in a format readable by Java-powered applications, corporations give users the platform-neutral access to the data they need.

When customers are running on the Java Platform, companies can take advantage of the interactivity of the Internet by moving employee tasks out to customers. Companies can reduce time spent on order-entry by having customers fill in order-entry forms themselves on Web pages. This is more practical than previously possible, because the customer can now be on any operating system.

Applets and Applications

The Java Platform enables developers to create two different kinds of programs:

- **Applets** are programs that require a browser to run. The `<applet>` tag is embedded in a Web page and names the program to be run. When that page is accessed by a user, either over the Internet or corporate intranet, the applet automatically downloads from the server and runs on the client machine. Because applets are downloaded, they tend to be designed small or modular, to avoid large download times.
- **Applications** are programs that do not require a browser to run—they have no built-in downloading mechanism. When an application is called, it runs. In this way, applications are just like programs in other languages. They can perform traditional desktop tasks, such as that done with a word processor, spreadsheet or graphics application. Like an applet, an application requires the Java Platform for it to run; however the Platform can be available as a

separate program, can be embedded directly within the underlying operating system, or presumably can be embedded in the application itself.

While applets and applications have different means of being invoked, for the most part they have the same access to a wide range of language capabilities. For example, either an applet or an application can access a host database, retrieve the data it needs, do local data processing, and store the results back to the host.

However, an applet requires a network to run, while an application does not. Applications have greater freedom in that they have full access to system services. For example an application, unlike an applet, can have normal read and write access to files on any disk. Since an applet can potentially be downloaded from an untrusted Web page, it is restricted from having read or write access to any file system except the server from which it came. This constraint will be relaxed when applets can be marked with digital signatures, allowing the end-user to be assured that it has been downloaded unaltered from a trusted source. Where local file storage is required, currently an application is required.

Where Will the Java Platform Be Deployed?

The progression towards ubiquity has great momentum, moving in three stages from browsers, to desktop, workstation and network operating systems, and finally to embedded devices.

First, the Java Base Platform is currently embedded in the most widely used Internet browser, Netscape Navigator™, and will soon be in Microsoft Internet Explorer. It is, or will become, available in other browsers, such as HotJava™.

Second, the Java Base Platform will soon be embedded in all leading desktop, workstation, and network operating systems—see Figure 1. Being available on the combination of Microsoft Windows, Macintosh, OS/2, and UNIX computers, the Java Base Platform will have an installed base as large as these platforms combined. By targeting this platform, developers will tap into the new, exploding market for Web and intranet applications without being tied to any particular hardware or operating system environment. The Java Platform will become the platform for all network- and Web-based computing.

Third, with the JavaChip™ family of integrated circuits, the platform will be available in a wide range of consumer and industrial embedded devices such as dedicated Network Computers, set-top boxes, printers, copiers and cellular phones.

Operating Systems That Embed the Java Base Platform

Windows	
• Microsoft Corporation	Windows 95, Windows NT
• International Business Machines	Windows 3.1
Macintosh	
• Apple Computer, Inc.	MacOS
OS/2	
• International Business Machines	OS/2
Unix	
• Hewlett Packard Corp.	HPUX
• Hitachi, Ltd.	Hitachi OS
• International Business Machines	AIX
• Silicon Graphics, Inc.	Irix
• SunSoft, Sun Microsystems, Inc.	Solaris™
• The Santa Cruz Operation, Inc. (SCO)	UnixWare
• Tandem Computers	Non-Stop Kernel
Network OS	
• Novell, Inc.	NetWare 4.0
Mainframe OS	
• International Business Machines	MVS

Figure 1. Companies licensing the Java Base Platform to embed in operating systems.

JavaChip™ Family

JavaSoft is working with Sun Microelectronics™ to develop the picoJava™, microJava™, and UltraJava™ family of microprocessors. The first of these, picoJava, is actually a standard specification for the design of a microprocessor that supports the Java Virtual Machine; this design is available for licensing to chip manufacturers. This design is a new architecture that is not SPARC-based—it is optimized for the unique demands of Java, such as multithreading and garbage collection.

The microJava and UltraJava are actual chips being developed by Sun Microelectronics based on the picoJava design. These chips have the Java Virtual Machine and Java Embedded API implemented in silicon, and vary in their application-specific I/O, memory, communications and control functions. The JavaOS can run in RAM on JavaChip. The JavaChip family enables the Java Virtual Machine to run in the most efficient, cost-effective manner, bringing high performance to dedicated Java-powered devices, such as Network Computers.

JavaOS™

JavaOS is an operating system that implements the Java Base Platform for running Java-powered applets and applications. As such, it implements the Java Virtual Machine, Java Embedded API, and the underlying functionality for windowing, networking and file system.

JavaOS is designed for Network Computers, consumer devices, and network devices for embedded applications, such as printers, copiers and industrial controllers. These devices will have instant turn-on, no installation setup, no system administration, and, when on a network, can be automatically upgraded.

JavaOS will be widely ported to a range of microprocessors, including the JavaChip family. When JavaOS runs on a JavaChip, the microprocessor's silicon Java Virtual Machine is used.

A Word About the Java Language[*]

The Java Language is the means for a developer to write source code. Applets and applications written in the Java Language compile to a form that runs on the Java Platform.

When developers write source code in the Java Language, this code can call APIs defined in the Java Base API, Java Standard Extensions API, or a new API defined in the source code itself. At runtime, all three kinds of APIs have equal standing, and are not distinguished on the basis of their source. Compiling the source with the Java Compiler generates bytecodes that are executed on the Java Platform.

As professional programming languages go, the Java Language is simple, yet flexible and powerful. It is object-oriented (with single inheritance), statically

* This section and the Virtual Machine section are borrowed heavily from the keynote address given by Bill Joy, Founder and Vice President of Research at Sun Microsystems, to Internet World on April 30, 1996.

typed, multithreaded, dynamically linked, and has automatic garbage collection.

Its syntax is based on C and C++, so those programmers can pick it up quite easily. There's less redundancy which means developers should be able to more easily read someone else's code. For example, the Java Language has no user-defined operator overloading, as is found in C++.

The Java Language gives developers the ability to do three different kinds of programming in one language. Like the symbolic programming language Smalltalk, the Java Language is object-oriented, has dynamic linking, and has a class hierarchy with single inheritance. For numeric programming, the Java Language has platform-independent data types, array bounds-checking, and well-defined IEEE arithmetic. These capabilities provide good grounding for writing stable numerical algorithms that give repeatable results. For systems programming, expressions, statements, and operators in the Java Language are in most cases the same as in the C language.

The Java Language encourages catching bugs early, during development, before the software is released. It does this by strong data typing, automatic garbage collection, array bounds checking, lack of automatic type coercion, and the lack of the pointer data type. These safeguards help in the age of the Internet, where developers are deploying software very rapidly.

The Java Language has multithreading built in, with a strong model of how thread-critical code can be synchronized to avoid race or timing problems. With the growth of multiprocessing and a decrease in processor costs, the Java Language is poised to enable a new generation of concurrent applications and services.

Exception and thread mechanisms are integrated with the language and its type system. In addition, the language includes dynamic linking of subclasses with methods that override or add functionality at runtime. In other environments, these features have typically been arcane and complicated system services. There is a great simplicity and advantage to having these facilities in the language and therefore portable between platforms. The language also defines what binary compatibility is, by defining a class (`.class`) file format, which includes the instructions for the Java Virtual Machine in the form of bytecodes.

A Look Inside the Java Platform

The Java Platform has two main parts, the Java Virtual Machine and the Java API, as shown in Figure 2.

- **Java Virtual Machine** - The Java Virtual Machine is a "soft" computer that can be implemented in software or hardware. It's an abstract machine designed to be implemented on top of existing processors. The porting interface and adapters enable it to be easily ported to new operating systems without being completely rewritten.
- **Java API** - The Java API forms a standard interface to applets and applications, regardless of the underlying operating system. The Java API is the essential framework for application development. This API specifies a set of essential interfaces in a growing number of key areas that developers will use to build their Java-powered applications.
  - **The Java Base API** provides the very basic language, utility, I/O, network, GUI, and applet services; OS companies that have licensed Java have contracted to include them in any Java Platform they deploy.
  - **The Java Standard Extension API** extends the capabilities of Java beyond the Java Base API. Some of these extensions will eventually migrate to the Java Base API. Other nonstandard extension APIs can be provided by the applet, application, or underlying operating system. As each new extension API specification is published, it will be made available for industry review and feedback before it is finalized.

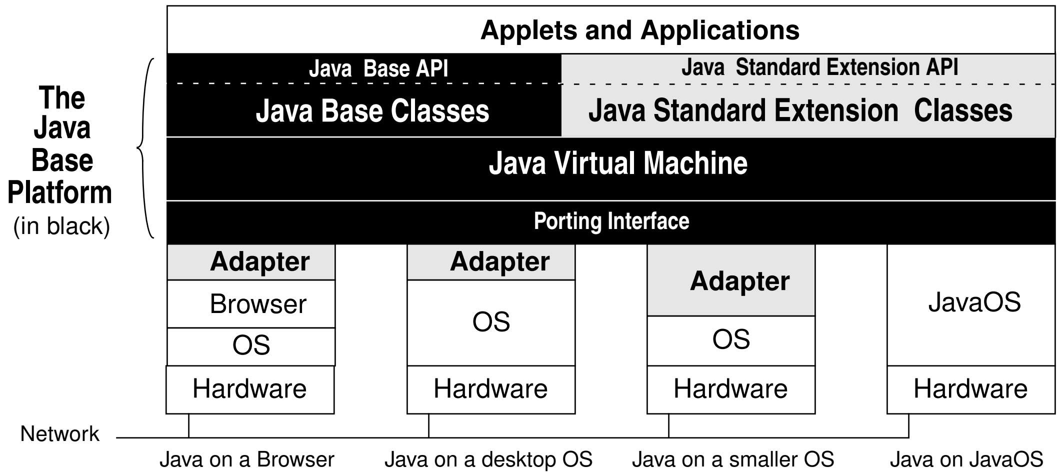

Figure 2. The Java Base Platform is uniform across all operating systems.

In this figure, the Java Base Platform is the part shown in black, including the blocks labeled Adapter. The Java API includes both the Java Base API and Java Standard Extension API. The classes are the implementation of that API. The Java Virtual Machine is at the core of the platform. The Porting Interface lies between the Java Virtual Machine and the operating system (OS) or browser. This Porting Interface has a platform independent part (shown in black) and a platform-dependent part, shown as Adapters. The OS and JavaOS provide the window, filing and network functionality. Different machines can be connected by a network, as shown.

The Java API framework is open and extensible. Specifications for each interface are being developed by industry-wide specialists in each area. Forthcoming specifications will be published and open to industry review. Implementations of the API specifications will be available from JavaSoft and others throughout the industry. In today's environment of rapid innovation, the Java API framework allows that innovation to easily exist as extensions to the Java Platform.

The API is organized by groups, or sets. Each of the API sets can be implemented as one or more packages (namespaces). Each package groups together a set of classes and interfaces that define a set of related fields, constructors, and methods.

Java Virtual Machine

The Java Virtual Machine is key to the independence of the underlying operating system and hardware—it is a platform that hides the underlying operating system from Java-powered applets and applications. And it's very easy to port the Virtual Machine to a browser or another operating system.

In addition, the Virtual Machine defines a machine-independent format for binary files called the class (`.class`) file format. This format includes instructions for a virtual computer in the form of bytecodes. The bytecode representation of any Java Language program is symbolic in the sense that offsets and indexes into methods are not constants, but are, instead, given symbolically as string names. The first time a method is called, it is searched by name in the class file format, and its offset numeric value is determined at that time for quicker access at subsequent lookups. Therefore, any new or overriding method can be introduced late at runtime anywhere in the class structure, and it will be referred to symbolically, and properly accessed without breaking the code.

The bytecodes are a high-level representation of the program so that optimization and machine code generation (via a just-in-time compiler) can be performed at that level. In addition, garbage collection can occur inside the Virtual Machine, since it holds variables in stacks in the Java Platform address space.

Java Base API

By targeting the APIs in the Java Platform, developers are assured that their applications will run everywhere. (See "The Embedded Java Platform" on page 7 for the single possible exception—dedicated, constrained, embedded systems, such as set-top boxes, printers, copiers, and cellular phones.)

Currently, the Java Base API is defined to be the Java Applet API, described in the next section. Over time, as the platform develops, this base will grow, as some of the Standard Extension API migrate into the Java Base API. The Java Base API is also known as the Java Core API.

Java Applet API

The Java Applet API, also known as the Java Base API, defines the basic building blocks for creating fully functional Java-powered applets and applications. It includes all the classes in the `java` package: `java.lang`, `java.util`, `java.io`, `java.net`, `java.awt`, and `java.applet`. (Notice that the Java Applet API is the full set of APIs available in version 1.0.2 of the Java Development Kit from JavaSoft.)

Java Standard Extension API

This section describes the Java Standard Extension API. These extensions are "standard" in that they form a published, uniform, open API that anyone can implement. Once defined, they can be added to, but, to maintain backward compatibility, not changed in a way that calls to them would fail. Over time, new extensions will be added. In addition, some of these extensions will migrate into the Java Base API.

The current plan for this migration is shown in the following table:

Migrate to Java Base API	**Remain as Java Standard Extension**
Java 2D	Java 3D
Audio	Video, MIDI
Java Media Framework	Java Share
Java Animation	Java Telephony
Java Enterprise	Java Server
Java Commerce	Java Management
Java Security	

An implementation should do whatever is appropriate for the platform it runs on, within its hardware constraints. For example, desktop operating systems that have access to a speaker will produce audio; however, a mainframe or network operating system that has no speaker is permitted to do the equivalent of a no-op or some other well-defined behavior, such as throw an exception.

Java Security API

The Java Security API is a framework for developers to easily and securely include security functionality in their applets and applications. This functionality includes cryptography, with digital signatures, encryption and authentication.

Java Security includes an abstract layer that applications can call. That layer, in turn, makes calls to Java Security packages that implement the actual cryptography. This allows third-party developers specializing in providing cryptographic functionality to write packages for Java Security. Java Security also includes system support for key management, including a secure database, certificate facilities, and so on.

This architecture provides for replaceable, upgradable security. Whenever a stronger algorithm or a faster implementation becomes available, modules can be replaced in the platform, in a manner completely transparent to applications.

Java Media API

The Java Media API defines the multimedia classes that support a wide range of rich, interactive media on and off the Web, including audio, video, 2D, 3D, animation, telephony, and collaboration. An extensible Media Framework provides for common control and synchronization of all time-based media (audio, video, animation, video teleconferencing) as well as for filters and processors.

The Java Media API is composed of several distinct components, each associated with either a specific type of media (audio, video, 2D, 3D), or a media-related activity (animation, collaboration, telephony). Collectively, these interfaces provide Java Language programmers with the ability to handle a wide variety of different media types within their applications and applets.

The Java Media API is highly extensible. The API accommodates today's large and ever-changing suite of media transports, containers, and encoding formats, and allows the addition of new media-related functionality as they become available.

JavaSoft has been working with a group of industry-leading companies to establish the standards for Java Media: Adobe®, Apple, Intel, Macromedia, Netscape, SGI, and Sun Microsystems.

The components of the Java Media APIs are as follows.

- **Java 2D API** - Provides graphics and imaging capabilities beyond those available with the Java Applet API. The 2D API allows the creation of high-quality, platform-independent graphics including line art, text, and images in a single model that uniformly addresses color, spatial transforms and compositing. It also provides an extension mechanism to support a wide array of different presentation devices (such as displays and printers), image formats, image encodings, color spaces, and compositors.
- **Java Media Framework API** - Handles traditional time-critical media, such as audio, video and MIDI. The framework provides a common model for timing, synchronization, and composition, which can be applied to media components to allow them to interoperate. It is designed to handle streaming data, live or stored, compressed or raw, as well as from sampled audio and video streams.
  - **Video API** - Accommodates both streaming and stored video sources. It defines basic data formats and control interfaces.

- **Audio API** - Supports sampled and synthesized audio. It includes a specification for 3D spatial audio, and accommodates both streaming and stored audio sources.
- **MIDI API** - Provides support for timed-event streams. It uses the Media Framework for synchronization with other activities, and for an extensibility mechanism for new synthesizers and effects.

- **Java Animation API** - Supports traditional 2D animation of sprites, with stacking order control. It makes use of 2D interfaces for compositing and the Media Framework for synchronization, composition, and timing.
- **Java Share API** - Provides the basic abstraction for live, two-way, multi-party communication between objects over a variety of networks and transport protocols. The API enables synchronization and session management, and allows sharing of both "collaboration-aware" and "collaboration-unaware" applets.
- **Java Telephony API** - Unifies computer/telephony integration. It provides basic functionality for control of phone calls: 1st-party call control (simple desktop phone), 3rd-party call control (phone call distribution center), teleconferencing, call transfer, caller ID, and DTMF decode/encode.
- **Java 3D API** - Provides high-performance, interactive, 3D graphics support. It supports VRML, and has a high-level specification of behavior and control of 3D objects. The 3D API simplifies 3D application programming and provides access to lower level interfaces for performance. The 3D API is closely integrated with Audio, Video, MIDI, and Animation areas.

Java Enterprise API

The Enterprise classes connect Java-powered applications to enterprise information resources. There are currently three groups for connectivity: JDBC™ (Java Database Connectivity), Interface Definition Language, and Remote Method Invocation.

- **JDBC™** (Java Database Connectivity) is a standard SQL database access interface. It provides Java programmers with a uniform interface to a wide range of relational databases, and also provides a common base on which higher level tools and interfaces can be built. Partners such as Intersolv, Visigenic, and a dozen other database-connectivity vendors are providing JDBC drivers in the next few months for dozens of DBMSs, including Oracle, Sybase, and Informix. These database companies, and leading tool vendors, such as Symantec and Borland, have already endorsed the JDBC API and are developing products using JDBC.

The JDBC API defines classes to represent constructs such as database connections, SQL statements, result sets, and database metadata. JDBC allows a Java-powered program to issue SQL statements and process the results.

In conjunction with JDBC, JavaSoft is releasing a JDBC-ODBC bridge implementation that allows any of the dozens of existing Microsoft ODBC database drivers to operate as JDBC drivers. The JDBC-ODBC bridge can run on the server rather than client side using a JDBC driver that translates to a DBMS-independent network protocol.

- **Interface Definition Language** (IDL) is a language-neutral way to specify an interface between an object and its client when they are on different platforms. This IDL component provides a mapping of its methods, packages, and other features to IDL operations and features.
- **Remote Method Invocation** (RMI) lets programmers create Java objects whose methods can be invoked from another virtual machine. RMI is analogous to a remote procedure call (RPC) in the non-object world.

Java Commerce API

The Java Commerce API brings secure purchasing and financial management to the Web.

World wide purchases of retail and wholesale goods and services during 1994 totaled 4.6 million dollars, 13% of which was spent remotely via catalogs, television, and various public and private communications networks. As this remote commerce migrates to the Internet, a standard framework within which to conduct these transactions becomes increasingly important. This is especially true as a rapidly increasing number of players vie for position to provide such facilities as electronic currency, online shopping malls, digital signature verification, and financial analysis.

The initial component of the Java Commerce API is the Java Wallet, which defines and implements a client-side framework for conducting network-based commerce. Think of Java Wallet as an empty wallet ready to hold credit cards and cash, and a blank ID card ready to be filled in with personal information.

The Java Wallet provides:

- Storage of personal information about:
  - The shopper (such as name, billing address, and shipping address)
  - Payment instruments (such as credit cards, debit cards, and electronic cash)
  - Details of every purchase transaction (such as date and time, item descriptions, quantities, and dollar amounts)
- Support for two new types of signed applets
  - Payment cassettes, which implement a specific payment protocol, such as the Secure Electronic Transaction (SET) supported by Visa and MasterCard
  - Service cassettes, which implement value-added services, such as budgeting and financial analysis
- Extensibility to install new payment and service cassettes dynamically by downloading them from the network
- Strong cryptographic services that make it possible to implement all of the facilities described above in a secure environment

Java Server API

Java Server is an extensible framework that enables and eases the development of a whole spectrum of Java-powered Internet and intranet servers. The framework API contains server-side class libraries for server administration, access control, and dynamic server resource handling. The framework also encompasses the Servlet API.

Servlets are platform-independent Java-powered objects, the server side counterpart of applets. Servlets can reside locally on the server, or be downloaded to the server from the net under security restrictions. Servlet examples range from simple HTTP servlets (efficient replacements of cgi-scripts) to more complex servlets using JDBC/ODBC that offer database connectivity.

Java Management API

The Java Management API is a collection of Java Classes that provide the building blocks for integrated management. It does this by providing a number of interfaces, classes, applets, and guidelines that facilitate the development of integrated management solutions.

The Java Management API is composed of several distinct components, each associated with an aspect of the total management space. Taken together the objects defined using the Java Management API will encapsulate distributed network, system, and service management components.

The components of the Java Management APIs are as follows:

- **Admin View Module** - The Admin View Module is an extension of the Java Abstract Window Toolkit (AWT) that is specifically designed for creating integrated management solutions. The Admin View Module classes are used to implement the user model that builds on the web browser hypertext style of navigation.
- **Base Object Interfaces** - The Base Object Interfaces support constructing objects that represent distributed resources and services that make up the enterprise computing environment. These interfaces allow developers to define abstractions that contain distributed attributes and methods, and persistent attributes.
- **Managed Notification Interface**s - The Managed Notification Interfaces provide the basic foundation from which more complex event-management services can be easily built. The model provides asynchronous event notification between managed objects or management applications providing the interfaces to an implementation of a basic event-dispatching service.
- **Managed Container Interfaces** - These allow managed objects to be grouped so that management applications can perform actions on a single group, instead of each instance. This permits management applications to scale upwards by allowing for multiple instances to be treated as one. Container interfaces are of two types: *Extensional,* which are constructed programmatically through simple add and remove methods, and *Intentional,* which store only the query to be executed—by using the Managed Data Interfaces—that generate the instances for the container.
- **Managed Data Interfaces** - These support mapping attributes of extensions to the Base Object Interfaces to a relational database. These interfaces are implemented on the appropriate subset of JDBC (Java Database

Connectivity). These interfaces support a number of commercially available relational database engines.

- **Managed Protocol Interfaces** - These implement the distribution and security capabilities for extensions of the Base Object Interfaces. These interfaces build up the Java Security APIs and Java Remote Method Invocation (RMI).
- **SNMP Interfaces** - These extend the Managed Protocol Interfaces to allow extensions of the Base Objects to contain information obtained from existing SNMP agents. By extending the Managed Protocol Interfaces, this allows SNMP information to be available to all users of the Java Management API.

The "Java Management API User Interface Style Guide" provides guidelines for developing interfaces for configuration and troubleshooting of the system, network, and service elements that make up the computing infrastructure.

Java Compile and Runtime Environments

A Java Language development environment includes both the compile-time and runtime environments, as shown in Figure 3. The Java Platform is represented by the runtime environment. The developer writes Java Language source code (`.java` files) and compiles it to bytecodes (`.class` files). These bytecodes are instructions for the Java Virtual Machine. To create an applet, the developer next stores these bytecode files on an HTTP server, and adds an `<applet code=`*filename*`>` tag to a Web page, which names the entry-point bytecode file.

When an end user visits that page, the `<applet>` tag causes the bytecode files to be transported over the network from the server to the end user's browser in the Java Platform. At this end, the bytecodes are loaded into memory and then verified for security before they enter the Virtual Machine.

Once in the Virtual Machine, the bytecodes are interpreted by the Intepreter, or optionally turned into machine code by the just-in-time (JIT) code generator, known more commonly as the JIT Compiler. The Interpreter and JIT Compiler operate in the context of the runtime system (threads, memory, other system resources). Any classes from the Java Class Libraries (API) are dynamically loaded as needed by the applet.

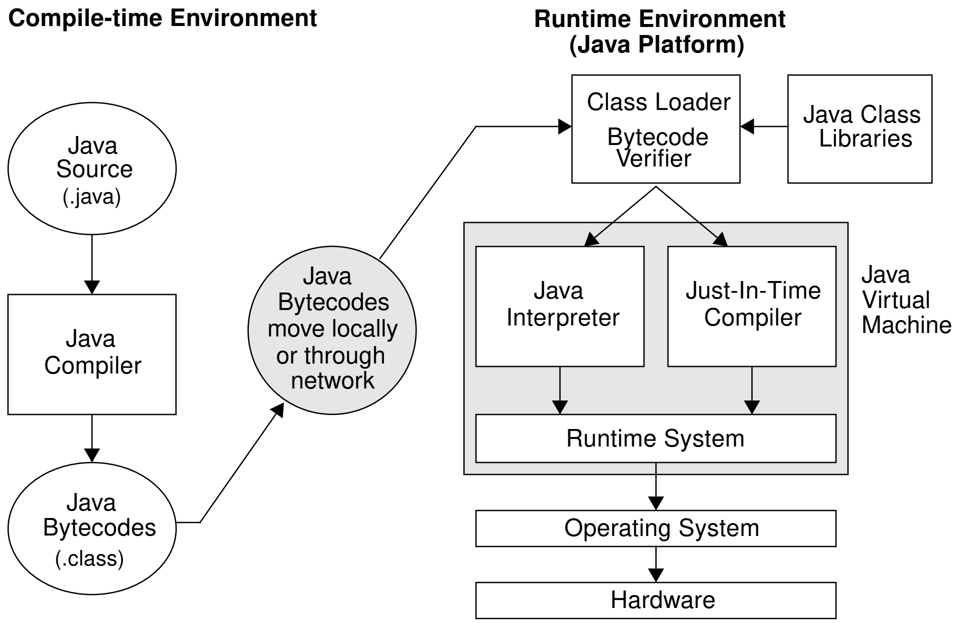

Figure 3. Source code is compiled to bytecodes, which are executed at runtime.

JavaSoft
A Sun Microsystems, Inc. Business

2550 Garcia Avenue
Mountain View, CA 94043
408-343-1400

For U.S. Sales Office locations, call:
800 821-4643
In California:
800 821-4642

Australia: (02) 844 5000
Belgium: 32 2 716 7911
Canada: 416 477-6745
Finland: +358-0-525561
France: (1) 30 67 50 00
Germany: (0) 89-46 00 8-0
Hong Kong: 852 802 4188
Italy: 039 60551
Japan: (03) 5717-5000
Korea: 822-563-8700
Latin America: 415 688-9464
The Netherlands: 033 501234
New Zealand: (04) 499 2344
Nordic Countries: +46 (0) 8 623 90 00
PRC: 861-849 2828
Singapore: 224 3388
Spain: (91) 5551648
Switzerland: (1) 825 71 11
Taiwan: 2-514-0567
UK: 0276 20444

Elsewhere in the world,
call Corporate Headquarters:
415 960-1300
Intercontinental Sales: 415 688-9000