5.2.1. JAVA (FRONT END)

Java is related to C++, which is a direct descendent of C. Much of the character of Java is inherited from these two languages. From C, Java derives its syntax. Many of Java’s object-oriented features were influenced by C++. In fact, several of Java’s defining characteristics come from—or are responses to—its predecessors. Moreover, the creation of Java was deeply rooted in the process of refinement and adaptation that has been occurring in computer programming languages for the past three decades. For these reasons, this section reviews the sequence of events and forces that led up to Java. As you will see, each innovation in language design was driven by the need to solve a fundamental problem that the preceding languages could not solve. Java is handles exception.

**About JAVA**

Java is a general purpose; Object Oriented Programming Language developed by the Sun Microsystems of USA in 1991. This language was initially called “*Oak*”.

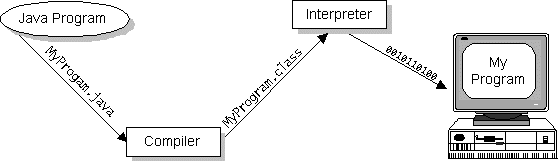
The primary motivation of Java was the need for a platform- independent (that is, architecture neutral) Language that could be used to create software. It is a platform- independent language that could be used to produce code that would run on a variety of CPUs under differing environments. This effort ultimately led to the creation of Java. With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called Java byte codes —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works. You can think of Java byte codes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it’s a development tool or a Web browser that can run applets, is an implementation of the Java VM.Java byte codes help make “write once, run anywhere” possible. You can compile your program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.

**3.3.1 Java language**

Java is no ordinary programming language. It inspires devotion, passion, exaltation and eccentricity. Java is a high-level programming language that is all of the following:

* Simple
* Object-oriented
* Distributed
* Interpreted
* Robust
* Secure
* Architecture-neutral
* Portable
* High-performance
* Multithreaded

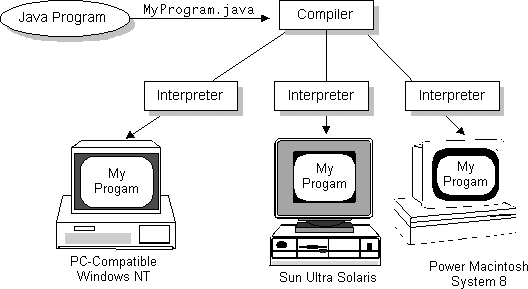
Java is also unusual in that each Java program is both compiled and interpreted. With a compiler, you translate a Java program into an intermediate language called Java byte codes--the platform-independent codes interpreted by the Java interpreter. With an interpreter, each Java byte code instruction is parsed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed. This figure illustrates how this works.



**Figure: 3.1 Working of Java**

Java byte codes can be considered as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware.

Java byte codes help make "write once, run anywhere" possible. The Java program can be compiled into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. For example, the same Java program can run on Windows NT, Solaris, and Macintosh.

[](file:///E:\Documents%20and%20Settings\user1.USER.001\tutorial\figures\getStarted\2comp.gif)

**Figure: 3.2 Execution of a Java program**

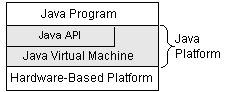
A platform is the hardware or software environment in which a program runs. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other, hardware-based platforms. Most other platforms are described as a combination of hardware and operating system.

The Java platform has two components:

* The Java Virtual Machine (Java VM)
* The Java Application Programming Interface (Java API)

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries (packages) of related components.

The following figure depicts a Java program, such as an application or applet, that's running on the Java platform. As the figure shows, the Java API and Virtual Machine insulates the Java program from hardware dependencies.



**Figure: 3.3 A Java program (Applet).**

As a platform-independent environment, Java can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring Java's performance close to that of native code without threatening portability.

**About Swings features and concepts**

This gives us the information you need to be able to use Swing components effectively. For example, it tells you how Swing programs display their GUIs, how they handle events such as mouse clicks, and how they can use features such as borders to help with layout. It ends with a discussion of how the features and concepts are used by a real program.

* [**Swing Components and the Containment Hierarchy**](file:///C:\Documents%20and%20Settings\Administrator\Desktop\JAVA\uiswing\overview\hierarchy.html)

Swing provides many standard GUI components such as buttons, lists, menus, and text areas, which you combine to create your program's GUI. It also includes containers such as windows and tool bars.

* [**Layout Management**](file:///C:\Documents%20and%20Settings\Administrator\Desktop\JAVA\uiswing\overview\layout.html)

Containers use layout managers to determine the size and position of the components they contain. Borders affect the layout of Swing GUIs by making Swing components larger. You can also use invisible components to affect layout.

### [Event Handling](file:///C:\Documents%20and%20Settings\Administrator\Desktop\JAVA\uiswing\overview\event.html)

Event handling is how programs respond to external events, such as the user pressing a mouse button. Swing programs perform all their painting and event handling in the event-dispatching thread.

### [Threads and Swing](file:///C:\Documents%20and%20Settings\Administrator\Desktop\JAVA\uiswing\overview\threads.html)

If you do something to a visible component that might depend on or affect its state, then you need to do it from the event-dispatching thread. This isn't an issue for many simple programs, which generally refer to components only in event-handling code. However, other programs need to use the invokeLater method to execute component-related calls in the event-dispatching thread.

## *5.2.2 What Can Java Do?*

Probably the most well-known Java programs are Java applets. An applet is a Java program that adheres to certain conventions that allow it to run within a Java-enabled browser.

However, Java is not just for writing cute, entertaining applets for the World Wide Web ("Web"). Java is a general-purpose, high-level programming language and a powerful software platform. Using the generous Java API, we can write many types of programs.

The most common types of programs are probably appletsand applications, where a Java application is a standalone program that runs directly on the Java platform.

**5.2.3 How does the Java API support all of these kinds of programs?**

With packages of software components that provide a wide range of functionality. The core API is the API included in every full implementation of the Java platform. The core API gives you the following features:

* **The Essentials**: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
* **Applets**: The set of conventions used by Java applets.
* **Networking**: URLs, TCP and UDP sockets, and IP addresses.
* **Internationalization**: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
* **Security**: Both low-level and high-level, including electronic signatures, public/private key management, access control, and certificates.
* **Software components**: Known as JavaBeans, can plug into existing component architectures such as Microsoft's OLE/COM/Active-X architecture, OpenDoc, and Netscape's Live Connect.
* **Object serialization**: Allows lightweight persistence and communication via Remote Method Invocation (RMI).
* **Java Database Connectivity (JDBC)**: Provides uniform access to a wide range of relational databases.
* Java not only has a core API, but also standard extensions. The standard extensions define APIs for 3D, servers, collaboration, telephony, speech, animation, and more.