A Short History of JAVA

Java goes back to 1991, when a group of Sun engineers, led **the Patric Naughton** and Sun Fellow (and all-around computer wizard) **James Gosling**, wanted to design a small computer language that could be used for consumer devices like cable TV switch boxes. Because these devices do not have a lot of power or memory. The language had to be small and generate very tight code. Also, because different manufacturer may choose different CPUs, it was important that the language not be tied to any single architecture. The project was code named "Green".

The requirement for small, tight and platform-neutral code led the team to resurrect the model that some Pascal implementations tried in the early days of PCs. **Niklaus Wirth**, the inventor of Pascal, pioneered the design of portable language that generated intermediate code for a hypothetical m/c. (These are often called virtual machines-hence the java virtual machine or JVM) This intermediate code could then be used on any machine that had the correct interpreter. The Green project engineers used a virtual m/c as well, so this solved their main problem.

The Sun people, however, come from a UNIX background, so they based their language on C++ rather than Pascal. In particular, they made the language object oriented rather than procedure oriented. But, as Gosling says in the interview, "All along, the language was a tool, not the end." Gosling decided to call his language "Oak" (presumably because he liked the look of an oak tree that was right outside his window at Sun). The people at Sun realized that Oak was the name of an existing computer language, so they changed the name Java. This turned out to be inspired choice.

In 1992, the Green project delivered its first product, called "*7". It was an extremely intelligent remote control. Unfortunately, no one interested in producing this at Sun, and the Green people had to find other ways to market their technology. However, none of the standard consumer electronics companies were interested. The group then bid on a project to design a cable TV box that could deal with new cable services such as video on demand.

The Green project (with new name of "First Person, Inc.") spent all of 1993 and half of 1994 looking for people to buy its technology-no one was found.

While all of this was going at Sun, the World Wide Web part of the Internet was growing bigger and bigger. The key to the web is the browser that translates the hypertext pages to the screen. In 1994, most people were using mosaic, a noncommercial web browser that came out of the supercomputing center at the university of Illinois 1993.

In the Sun World interview, Gosling says that in mid-1994, the language developers realized that "We could build a real cool browser. It was one of the few things in the client/server mainstream that needed some of the weird things we'd done: architecture neutral, real time, reliable, secure-issues that weren't terribly important in the workstation world. So we built a browser."

The actual browser was built by Patrick Noughton and Jonathan Payne and evolved into the Hot Java browser. The Hot Java browser was written in Java to off the power of Java.
But the builders also had in mind the power of what are now called applets, so they made the browser capable of executing code inside the pages. This "proof of technology" was shown at SunWorld '95 on May 23,1995 and inspired Java craze that continues today.

Sun released the first version of Java in early 1996. People quickly realized that Java 1.0 was not going to cut it for serious application development. Sure, you could use Java 1.0 to make nervous text applet that move text randomly around in a canvas. But you couldn't even print in Java 1.0. To beb blunt, Java 1.0 was not ready for prime time. Its successor version 1.1, filled in the most obvious gaps, greatly improved the reflection capability, and added a new event model for GUI programming. It was still rather limited, though

The big news of the 1998 Java One conference was the upcoming release of Java 1.2, which replaced the early toy like GUI and graphics toolkits with sophisticated and scalable versions that come a lot closer to promise of "Write Once Run Anywhere" than their predecessor. Three days after its release in December 1998, Sun's marketing department changed the name to the catchy "Java 2 Standard Edition Software Development Kit version 1.2".

Besides the "Standard Edition" 2 other editions were introduced: the "Micro Editions" for embedded devices such as cell phones and "Enterprise Edition" for server-side programming.

Version 1.3 and 1.4 of the Standard Edition are incremental improvement over the initial Java 2 release, with an ever- growing standard library, increased performance and of course, quite a few bug fixes.

What is JAVA?

JAVA is a high-level, general-purpose, concurrent, class-based pure Object Oriented programming language. JAVA is related to C and C++ but rather differently, with a no. of aspects of C & C++ omitted & a few ideas from the language included.

JAVA can be used to create two types of programs: Application & Applet.

- An Application is a program that runs on our computer, under the Operating System of that computer.
- An Applet is a program designed to be transmitted over the internet and executed by any JAVA compatible web browser.

Java - Originally called Oak by creator James Gosling, from the tree that stood outside his window, the programming team had to look for a substitute as there was another language with the same name. Java was selected from a list of suggestions. It came from the name of the coffee that the programmers drank.

Two Failure of Java

Java was designed by Sun Microsystems in the early 1990s to solve the problem of connecting many household machines together. This project failed because no one wanted to use it.

Then it was redesigned to work with cable TV. This project also failed because the cable companies decided to choose a competing system.

Success

When the World Wide Web became popular in 1994, Sun realized that Java was the perfect programming language for the Web. Early in 1996 (late 1995?) they released Java (previously named Oak) and it was an instant success! It was a success, not because of marketing, but because there was a great need for a language with its characteristics.

Team Members

James Gosling, Niklaus Wirth(the inventor of Pascal, Design JVM), the Patric Naughton and Jonathan Payne (Hot Java Browser designer)

Versions:

- Java 1.0 - 211 classes in 8 packages released May 1996.
- Java 1.1 - 477 classes in 23 packages released Feb 1997.
- Java 1.2/2.0 - 1,524 classes in 59 packages, released Dec 1998.
- Java 1.3 – 1840 classes released in 2000
- Java 1.4 - 2723 classes released in 2004
- Java 5.0 - 3269 classes in 166 packages released in 2004
- Java 6 -3777 classes released in 2006

Introduction

- ✓ Java technology is a high-level programming and a platform independent language. Java is designed to work in the distributed environment on the Internet.
- ✓ Java has a GUI features that provides you better "look and feel" over the C++ language, moreover it is easier to use than C++ and works on the concept of object-oriented programming model.
- ✓ Java enables us to play online games, video, audio, chat with people around the world, Banking Application, view 3D image and Shopping Cart. Java find its extensive use in the intranet applications and other e-business solutions that are the grassroots of corporate computing.
- ✓ Java, regarded as the most well described and planned language to develop applications for the Web.
- ✓ Java is a well known technology which allows you for software designed and written only once for an "virtual machine" to run on a different computers, supports various Operating System like Windows PCs, Macintoshes, and Unix computers.
- ✓ On the web aspect, Java is popular on web servers, used by many of the largest interactive websites. Java is used to create standalone applications which may run on a single computer or in distributed network. It is also be used to create a small application program based on applet, which is further used for Web page. Applets make easy and possible to interact with the Web page.

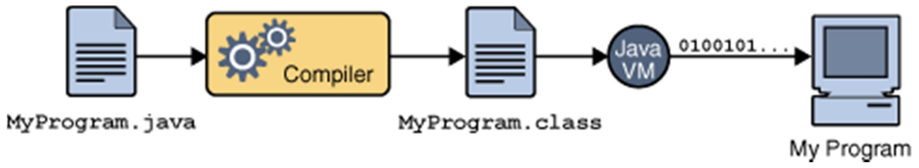
Java technology is both a programming language and a platform.

The Java Programming Language

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

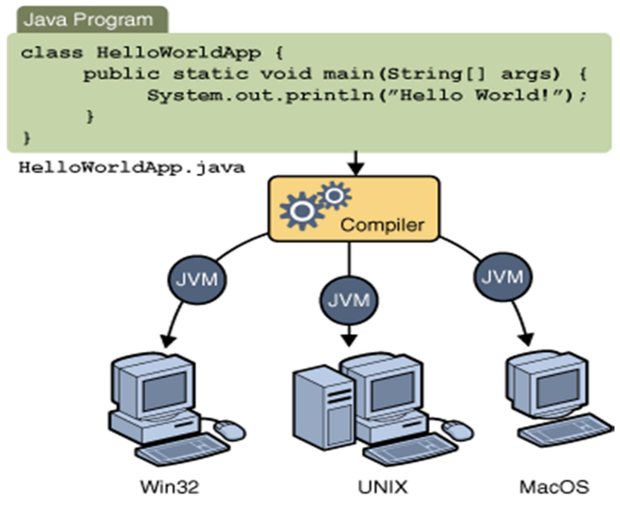
☐ Simple	☐ Architecture neutral
☐ Object oriented	☐ Portable
☐ Distributed	☐ High performance
☐ Multithreaded	☐ Robust
☐ Dynamic	☐ Secure
☐ Interpreted	☐ Network-Savvy

Each of the preceding buzzwords is explained in *The Java Language Environment* , a white paper written by James Gosling and Henry McGilton. In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the javac compiler. A .class file does not contain code that is native to your processor; it instead contains *bytecodes* — the machine language of the **Java Virtual Machine**[1] (Java VM). The java launcher tool then runs your application with an instance of the Java Virtual Machine.



An overview of the software development process.

Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris ™ Operating System (Solaris OS), Linux, or Mac OS. Some virtual machines, such as the Java HotSpot virtual machine, perform additional steps at runtime to give your application a performance boost. This include various tasks such as finding performance bottlenecks and recompiling (to native code) frequently used sections of code.



```
class HelloWorldApp {
     public static void main(String[] args) {
          System.out.println("Hello World!");
     }
}
```

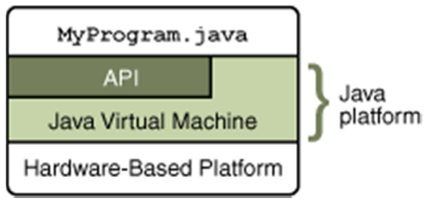
Through the Java VM, the same application is capable of running on multiple platforms.

The Java Platform

A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Microsoft Windows, Linux, Solaris OS, and Mac OS. Most platforms can be described as a combination of the operating system and underlying hardware. 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.
The Java platform has two components:

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

You've already been introduced to the Java Virtual Machine; it's the base for the Java platform and is ported onto various hardware-based platforms. The API is a large collection of ready-made software components that provide many useful capabilities. It is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section, What Can Java Technology Do? highlights some of the functionality provided by the API.



The API and Java Virtual Machine insulate the program from the underlying hardware.

As a platform-independent environment, the Java platform can be a bit slower than native code. However, advances in compiler and virtual machine technologies are bringing performance close to that of native code without threatening portability.

The terms "Java Virtual Machine" and "JVM" means a Virtual Machine for the Java platform.

The advantages of Java are as follows:

- Java is simple, easy to design , easy to write, and therefore easy to compile, debug, and learn than any other programming languages.
- Java is object-oriented, that is used to build modular programs and reusable code in other application.
- Java is platform-independent and flexible in nature. The most significant feature of Java is to run a program easily from one computer system to another.
- Java works on distributed environment. It is designed to work on distributed computing , Any network programs in Java is same as sending and receiving data to and from a file.
- Java is secure. The Java language, compiler, interpreter and runtime environment are securable .
- Java is robust. Robust means reliability. Java emphasis on checking for possible errors, as Java compilers are able to detect many error problem in program during the execution of respective program code.
- Java supports multithreaded. Multithreaded is the path of execution for a program to perform several tasks simultaneously within a program. The java come with the concept of Multithreaded Program. In other languages, operating system-specific procedures have to be called in order to work on multithreading.

Java Technology Works

Java is a high-level programming language and powerful software platform. On full implementation of the Java platform gives you the following features:

- **JDK Tools**: The JDK tools provide compiling, Interpreter, running, monitoring, debugging, and documenting your applications. The main tools used are the Javac compiler, the java launcher, and the javadoc documentation tool.
- **Application Programming Interface (API)**: The API provides the core functionality of the Java programming language. It gives a wide collection of useful classes, which is further used in your own applications. It provides basic objects and interface to networking and security, to XML generation and database access, and much more.

- **Deployment Technologies**: The JDK software provides two type of deployment technology such as the Java Web Start software and Java Plug-In software for deploying your applications to end users.
- **Graphical User Interface Toolkits**: The Swing and Java 2D toolkits provide us the feature of Graphical User Interfaces (GUIs).
- **Integrated Libraries**: Integrated with various libraries such as the Java IDL API, JDBC API, Java Naming and Directory Interface TM ("J.N.D.I.") API, Java RMI, and Java Remote Method Invocation over Internet Inter-ORB Protocol Technology (Java RMI-IIOP Technology) enable database to access and changes of remote objects.

Java Technology Changes Our Life:

- **Easy to Start**: Since Java programming language is completely based on object-oriented language, it's easy very simple and easy to learn, especially for programmers already known with C or C++.
- **Easy to write code**: As compared to program metrics (class counts, method counts, and so on) tell us that a program written in the Java programming language can be four times smaller as compare to the same program written in C++.
- **Write better code**: The Java programming language encourages good coding practices, and manages automatic garbage collection which helps you avoid memory leaks. Based on the concept of object orientation, its Java Beans component architecture, and wide-range, easily extendible, flexibility and API can reuse existing, tested code and introduce fewer bugs.
- **Develop programs and Time Safer**: The Java programming language is easier and simpler than C++, as such, manages your development time upto twice as fast when writing in it. The programs will also require fewer lines of code.
- **Platform Independencies**: The program keep portable and platform independent by avoiding the use of libraries written in other languages.
- **Write Once and Used in any Java Platform** : Any Source code of Program are written in the Java programming language, that is compiled into machine-independent byte codes and run consistently on any platform of java.
- **Distribute software makes work easy** : Using Java Web Start software, users will be able to launch own applications with a single click on mouse. An automatic version check initially weather users are always up to date with the latest version of your software. If an update is available for it, the Java Web Start software will automatically update their installation.

Features of JAVA

1. **Simple**

JAVA provides a System that could be programmed easily without a lot of esoteric training and which leveraged today's standard practice. We designed Java as closely to C++ as possible in order to make the System more comprehensible. Java omits many rarely used, poorly understood, confusing features of C++. Some of these are:

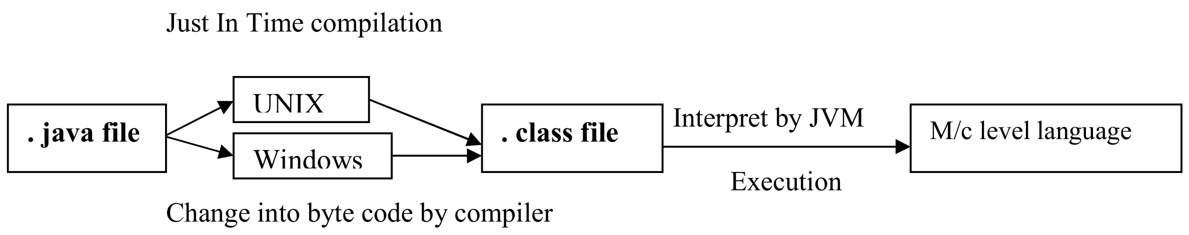
S. No.	**Features**	**C++**	**JAVA**
1	Multiple Inheritance	Yes	No
2	Pointers	Yes	No
3	Including Header Files	Yes	No
4	Structures	Yes	No
5	Unions	Yes	No
6	Operator Overloading	Yes	No
7	Virtual Base Classes	Yes	No
8	Semicolon at the end of the Class	Yes	No
9	Destructors	Yes	No
10	Friend functions	Yes	No
11	goto & delete keyword	Yes	No

2. **Platform Independency**

The main objective of java is "Compile once run anywhere".

Architecture Neutral (Version Independency)

The java compiler does this by generating byte code instructions, which have nothing to do with particular computer architecture. They are designed to both easy to interpret on any machine and easily translated into native machine code.



Byte Code - The compiler generates an architecture neutral object file format i.e. class file written with special code according to the operating system called **byte code (compiled code).** The byte code is only understandable by the JVM (Java Virtual Machine). It is neither a human understandable nor machine level language.
The compiled code is executable on many processors, given the presence of the JAVA runtime system. The use of byte code enables the Java run-time system to execute programs much faster than you might expect.

Java Virtual Machine: - *Byte code* is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the ***Java Virtual Machine*** **(*JVM*).** That is, in its standard form, the JVM is an ***interpreter*** *for byte code.*

While we work with C, C++ and Visual Basic like languages, whenever we try to compile our programs, compiler will 1st check out what is the platform on which it is running & then will create a **.exe** file, which is written in m/c level language. This **.exe** file is accepted for similar platform only. This is what we call as **platform dependency** in which the program can be executed on a system, which is similar to the system on which it is executed.

In contrast to this when we try to compile the JAVA program compiler will not consider the platform and will create a platform independent **.class** file. This file will create according to the JAVA Byte Code Standard which intermediate level language.

Whenever we try to compile the same java program on different platform, who will get the same **.class** file irrespective of platform. Because of this reason java compiler is called platform independent.

When we try to run the program the system is unable to understand the Byte Code. Because of this Java has introduced **Java Virtual Machine (JVM)**, which is capable of reading the and translate it into m/c level code at run time. JVM will check what is the platform on which the program is running and will translate the Byte Code into m/c level one and deliver the command to the operating system. This O.S. executes the command given by the JVM and provides the output.

Java will considering the platform on which it is used and will translate the byte code according to underlining platform. Because of this JVM is treated as Platform dependent.

3. <u>Object-Oriented</u>

Object Oriented design is a technique for programming that focuses on the data (Object) and on interfaces to that object. The object-oriented facilities of JAVA are essentially those of C++.

The major differences between JAVA and C++ lies in multiple inheritance, which JAVA replaced with simpler concept interfaces and JAVA Meta class Model. The reflection mechanism and Object Serialization feature make it much easier to implement persistent objects & GUI builders that can integrate of the Shelf Components.

4. <u>Distributed</u>

Java has an extensive library of routines for copying with TCP/IP protocols like HTTP & FTP. JAVA applications can open and access objects across the Net via URLs with the same as when accessing a local file system.

The networking capabilities of JAVA to be both strong and easy to use. The Remote Method Invocation (RMI) mechanism enables communication between distributed objects.The J2EE Support very large scale distributed applications.

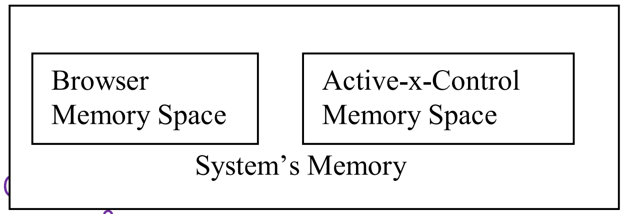
5. <u>Robust</u>

JAVA is intended for writing programs that must be reliable in variety of ways. JAVA puts a lot of emphasis on early checking for possible problems, later dynamic (run-time) checking and eliminating situations that are error-prone.

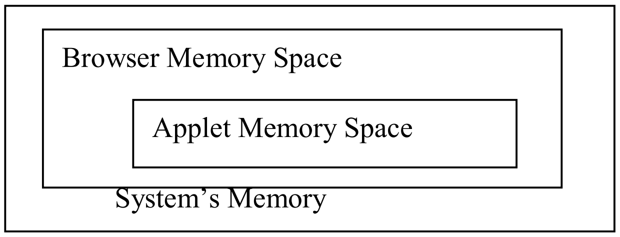
To gain reliability, Java restricts you in a few key areas, to force you to find your mistakes early in program development. At the same time, Java frees you from having to worry about many of the most common causes of programming errors. Because Java is a strictly typed language, it checks your code at compile time.

6. <u>Security</u>

While we trying to view a web page that contains an active-x-control the html code in association with active-x-control will be downloaded onto our system, while displaying the web page content, the browser will try to execute the active-x-control for this operation, O.S. will allocate some memory space for the active-x-control is directly coming in touch with the Systems memory space. Because of this an active-x-control can do anything on your system. Because of this reason active-x-controls are highly unsecured.



When we come across web based java program it offering applets as replacement for active-x-controls. Whenever we download a web page that contain an applet, memory space will be given for applet which falls within the memory space of the browser application. Because of this an applet can access only browser based data and non of the client's data. Because of this the client's system will be secured. This approach is called **Sand-Box Model**.



7. Multithreading

Process: program at executed (system based).
Multi processing: running more then one process on the same system.
Parallel processing: single process running on multiple systems.
Task: a peace of work.
Multi tasking: if the process is doing more then one task at a time.

Thread is nothing but a **separate flow of execution within a task**. If we are handling a task with the help of more then one thread i.e. called **Multithreading**.

While we work with other languages they are going to allows us to work with implicitly created threads only. But in java be a developer we can create any no. of threads according to the requirement. Because of this we can implement parallel under a task level. This will increase efficiency of the application.

8. Garbage Collection.

While working with most of the languages, dynamic memory allocation is acceptable. But at the same time the was forced to reallocate the memory also. If we are not doing so, data of one application is available foe another application, called **Memory Leakage.**

When we run any java program dynamically memory will be allocated. In the background of your application Garbage collector will be running, which is going to take care of freeing the memory space, no more useful for the program, there is no need to bother about memory freeing.

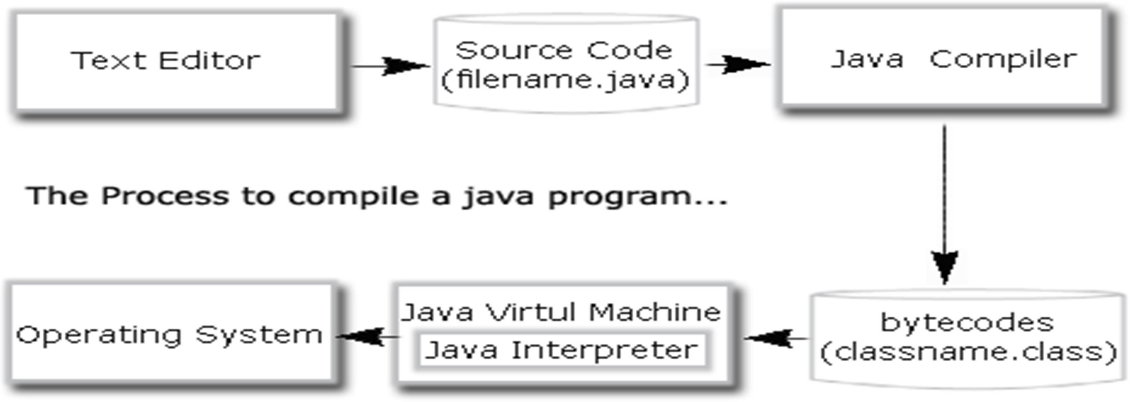
9. Exception Handling:

Because of the implicit type casting, no pointers, efficient run time error handling java is enabling developing the programs much more efficiently when compare to the others.

Java Tools:

1. **javac (Java Compiler)**

   A Java Compiler javac is a computer program or set of programs which translates java source code into java byte code. The output from a Java compiler comes in the form of Java class files (with .class extension). The java source code contained in files end with the .java extension. The file name must be the same as the class name, as classname.java. When the javac compiles the source file defined in a .java files, it generates bytecode for the java source file and saves in a class file with a .class extension.


Text Editor
Source Code (filename.java)
Java Compiler
The Process to compile a java program...
bytecodes (classname.class)
Java Virtul Machine
Java Interpreter
Operating System

Once the byte code is generated it can be run on any platform using Java Interpreter (JVM). It interprets byte code (.class file) and converts into machine specific **binary code**. Then JVM runs the binary code on the host machine.

How To Use:
Step 1. C:>jadk 1.6>javac java sourcefile name with .java extension
Step 2. C:>jadk 1.6>java java sourcefile name without .java extension

2. java (Java Interpreter)

Java interpreter translates the **Java bytecode into the code that can be understood by the Operating System**. Basically, A Java interpreter is a software that implements the Java virtual machine and runs Java applications.

3. Jdb (Java Debuger)

Java debugger helps in finding and the fixing of bugs in Java language programs. The Java debugger is denoted as **jdb.** It works like a command-line debugger for Java classes.

4. Javah (header file include)

In Java programming we need to implement some **native methods**. You must be wondering about what's a native method. **Native methods are the only way to use any system features not provided by the Java Virtual Machine.**
To implement these methods **Javah** generates C header and source files that are used by C programs to reference an Object's instance variables from native source code. The name of the header file and the structure declared within it are derived from the name of the class.

By default **javah** creates a header file for each class listed on the command line and puts the files in the current directory. As stated above the name of the header file is derived from the name of the class. If any class inside the package is passed to javah, the package name gets prepended to both the header file name and the structure name.

5. javadoc (documentation creator)

This tool is used to generate API documentation into HTML format from Java source code. It is interesting to know that Javadoc is the industry standard for documenting Java classes.

Javadoc is a program that is already included in JDK. We can use Javadoc to run over the source code to produce documentation of our classes in the HTML files . We have to tag our code with by using some comment formats to use javadoc tag on our source code. For instance Javadoc comments looks like this:

NOTE : To start a Javadoc comments use /** to start, and */ to end, and use tags such as @param,

@return, and @exception in between to describe the workings of a method.

The format is given below to use the Javadoc comments:

```
/**
* Summary of the sentence.
*  information about the
* program, class, method or variable
* then the comment, using as many lines
* as necessary.
*
*/
```

To document source code developers use certain commenting styles and Javadoc tags. A Java block comment starting with /** will begin a Javadoc comment block This comment block will be included in the HTML.

6. Java Applet viewer

Applet viewer is a command line program to run Java applets. It is included in the SDK. It helps you to test an **applet** before you run it in a browser.

An **applet** is a special type of application that's included as a part of an **HTML** page and can be stored in a web page and run within a web browser. The applet's code gets transferred to the system and then the ***Java Virtual Machine*** (JVM) of the browser executes that code and displays the output.. So for running the applet, the browser should be Java enabled. To create an applet, we need to define a class that inherits the Applet.

The difference in using the applet viewer and the web browser to run the applet is that the applet viewer only deals with the applet code not the HTML cod i.e. it doesn't display HTML code. So we should test our program in applet viewer and web browser to confirm its working.

The **applet viewer command** connects to the documents or resources designated by urls. It displays each applet referenced by the documents in its own window.

The syntax for the applet viewer is:

appletviewer Options URL

Where the URL specifies the location of the applet program and the Options argument specifies how to run the Java applet. We can use only one option -debug that starts the applet viewer in the Java debugger. Using this option we can debug an applet.

7. Javap (Java deassembler):

It is used to display the content of the predefined packages of java.

Syntax: c:>javap java.lang.String

TYPES OF STORAGE

It's useful to visualize some aspects of how things are laid out while the program is running, in particular how memory is arranged. There are six different places to store data:

1. **Registers**. This is the fastest storage because it exists in a place different from that of other storage: inside the processor. However, the number of registers is severely limited, so registers are allocated by the compiler according to its needs. You don't have direct control, nor do you see any evidence in your programs that registers even exist.
2. **The stack**. This lives in the general RAM (random-access memory) area, but has direct support from the processor via its *stack pointer*. The stack pointer is moved down to create new memory and moved up to release that memory. This is an extremely fast and efficient way to allocate storage, second only to registers. The Java compiler must know, while it is creating the program, the exact size and lifetime of all the data that is stored on the stack, because it must generate the code to move the stack pointer up and down. This constraint places limits on the flexibility of your programs, so while some Java storage exists on the stack— in particular, object references—Java objects themselves are not placed on the stack.
3. **The heap**. This is a general-purpose pool of memory (also in the RAM area) where all Java objects live. The nice thing about the heap is that, unlike the stack, the compiler doesn't need to know how much storage it needs to allocate from the heap or how long that storage must stay on the heap. Thus, there's a great deal of flexibility in using storage on the heap. Whenever you need to create an object, you simply write the code to create it using **new**, and the storage is allocated on the heap when that code is executed.
   Of course there's a price you pay for this flexibility: it takes more time to allocate heap storage than it does to allocate stack storage (that is, if you even *could* create objects on the stack in Java, as you can in C++).
4. **Static storage**. "Static" is used here in the sense of "in a fixedlocation" (although it's also in RAM). Static storage contains data that is available for the entire time a program is running. You can use the **static** keyword to specify that a particular element of an object is static, but Java objects themselves are never placed in static storage.
5. **Constant storage**. Constant values are often placed directly in the program code, which is safe since they can never change. Sometimes constants are cordoned off by themselves so that they can be optionally placed in read-only memory (ROM).
6. **Non-RAM storage**. If data lives completely outside a program it can exist while the program is not running, outside the control of the program. The two primary examples of this are *streamed objects,* in which objects are turned into streams of bytes, generally to be sent to another machine, and *persistent objects,* in which the objects are placed on disk so they will hold their state even when the program is terminated.
   The trick with these types of storage is turning the objects into something that can exist on the other medium, and yet can be resurrected into a regular RAM-based object when necessary. Java provides support for *lightweight persistence*, and future versions of Java might provide more complete solutions for persistence.

Example of Simple Java Program:

```
Line 1:    public class Example
Line 2:      {
Line 3          // Your program begins with a call to main().
Line 4:        public static void main(String args[])
Line 5:            {
Line 6:                  System.out.println ("This is a simple Java program.");
Line 7:            }
Line 8:      }
```

NOTE:

1. The Java compiler requires that a source file use the **.java** filename extension.
2. A source file is officially called a *compilation unit.* It is a text file that contains one or more class definitions.
3. The name of the source file must be the name of the class that contain main() function. e. g. For the above program source file name is **Example.java.**
4. That class must be **public** when the program has more than one class.

Compilation of the program:

To compile the **Example** program, execute the compiler, **javac**, specifying the name of the source file on the command line, as shown here:

```
C:\javap>javac Example.java
```

The **javac** compiler creates a file called **Example.class** that contains the byte code version of the program. To actually run the program, you must use the Java interpreter, called **java**. To do so, pass the class name **Example** as a command-line argument, as shown here:

```
C:\javap>java Example
```

When the program is run, the following output is displayed:

"This is a simple Java program."

Points to be Noted:

Line 1,2 & 8: **public class Example {**

- ✓ The **public** keyword is an *access specifier,* which allows the programmer to control the visibility of class members. In this case, it is optional. When the program has more than 1 class, then the class, that has main() function must be declared as **public.**
- ✓ The keyword **class** to declare that a new class is being defined. **Example** is an *identifier* that is the name of the class.
- ✓ The entire class definition, including all of its members, will be between the opening curly brace ({) and the closing curly brace (}).

Line 3: The next line in the program is the *single-line comment,* shown here:

// Your program begins with a call to main ().

This is the second type of comment supported by Java. A *single-line comment* begins with a // and ends at the end of the line.

Line 4: **public static void main (String args[])**

- When a class member is preceded by **public**, then that member may be accessed by code outside the class in which it is declared. In this case, **main( )** must be declared as **public**, since it must be called by code outside of its class when the program is started.

- The keyword **static** allows **main( )** to be called without having to instantiate a particular instance of the class. This is necessary since **main( )** is called by the Java interpreter before any objects are made.

- The keyword **void** simply tells the compiler that **main( )** does not return a value.

- This line begins the **main( )** method. this is the line at which the program will begin executing. All Java applications begin execution by calling **main( )**. (This is just like C/C++.)

- In **main( )**, there is only one parameter. **String args[ ]** declares a parameter named **args**, which is an array of instances of the class **String**. (*Arrays* are collections of similar objects.) Objects of type **String** store character strings. In this case, **args** receives any command-line arguments present when the program is executed.

Line 5 & 7: { and } denotes the starting and ending of the main() function.

Line 6: The output statement "**System.out.println ("This is a simple Java program.");** "
- **System:** Name of a class contained in java.lang package. System class contain 3 predefined objects: in, out, err.
- **System.out:** out is an abject of the System class. It refers to the Standard output stream (Visual Display Unit).

- **println:** Method that displays string (Messages) (Monitor).

White spaces: Java is a free-form language. This means that you do not need to follow any special indentation rules. In Java, whitespace is a space, tab, or newline.

Identifier: Identifiers are used for class names, method names, and variable names. An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the
underscore and dollar-sign characters. They must not begin with a number, lest they be confused with a numeric literal. Again, Java is case-sensitive, so **VALUE** is a different identifier than **Value**. Some examples of valid identifiers are:
AvgTemp count a4 $test this_is_ok

Invalid variable names include:
2count high temp Not/ok

Literals
A constant value in Java is created by using a *literal* representation of it. For example, here are some literals:
100 98.6 'X' "This is a test"

Left to right, the first literal specifies an integer, the next is a floating-point value, the third is a character constant, and the last is a string. A literal can be used anywhere a value of its type is allowed.

Comments
As mentioned, there are three types of comments defined by Java.
1. Single Line Comment: Java provides single line comment starting with "//" Operator and end with the line.
2. Multi Line comment: Like with C / C++ Languages, Java Support Multi line comment start with /* and ends with a */. E.g.
   /* this is
   a multi line comment */
3. Documentation comment: Java add new feature documentation comment. A comment start with /** and ends with */. Each /** ………… */ documentation comment contains free-from text followed by tags. A tag starts with an @, such as @author or @param.
   The first sentence of the free from text should be a summary statement. The **javadoc** utility automatically generates html pages tha extract these sentences.

Variables

It is the name of location where data will be stored. The variable is the basic unit of storage in a Java program. A variable is defined by the combination of an identifier, a type, and an optional initializer.

Declaration Of variable:

type identifier [= *value*][, *identifier* [= *value*] ...] ;

- The *type* is one of Java's atomic types, or the name of a class or interface.
- The *identifier* is the name of the variable.

```
E.g.        int a, b, c;                 // declares three ints a, b, and c.
            float f1,f2;                 //declares two floats f1 and f2.
```

Definitions of Variables:

E. g. int i;

```
    i=10;        //memory allocated for i=10
int d = 3, e, f = 5;         // declares three more ints, initializing d and f.
byte z = 22; // initializes z.
double pi = 3.14159; // declares an approximation of pi.
char x = 'x'; // the variable x has the value 'x'.
```

Special case: primitive types

There is a group of types that gets special treatment; you can think of these as "primitive" types that you use quite often in your programming. The reason for the special treatment is that to create an object with **new**—especially a small, simple variable—isn't very efficient because **new** places objects on the heap. For these types Java falls back on the approach taken by C and C++. That is, instead of creating the variable using **new**, an
"automatic" variable is created that *is not a reference*. The variable holds the value, and it's placed on the stack so it's much more efficient. Java determines the size of each primitive type. These sizes don't change from one machine architecture to another as they do in most languages. This size invariance is one reason Java programs are so portable.

Primitive type	**Size**	**Minimum**	**Maximum**	**Default Value**	**Wrapper type**
boolean				FALSE	Boolean
char	16- bit	Unicode 0	Unicode 216-1	'\u0000' (null)	Character
byte	8 bit	-128	127	(byte)0	Byte
short	16 bit			(short)0	Short
int	32 bit			0	Integer
long	64 bit			0L	Long
float	32 bit			0.0f	Float
double	64 bit			0.0d	Double
void					Void

Default values for primitive members:

When a primitive data type is a member of a class, it is guaranteed to get a
default value if you do not initialize it:

Primitive type	**Default**
boolean	**false**
char	**'\u0000' (null)**
byte	**(byte)0**
short	**(short)0**
int	**0**
long	**0L**
float	**0.0f**
double	**0.0d**

Operators

Java provides a rich operator environment. Most of its operators can be divided into the following four groups: arithmetic, bitwise, relational, and logical. Java also defines some additional operators that handle certain special situations.

1. **Arithmetic Operators**

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:
+, -, *, /, %, +=, -=, *=, /=, %=, - -

2. **Increment and Decrement operator**

The ++ and the - - are Java's increment and decrement operators. The increment operator increases its operand by one. The decrement operator decreases its operand by one.

For example, this statement: x = x + 1; can be rewritten like this by use of the increment operator: x++; Similarly, this statement: x = x - 1; is equivalent to x—;

These operators are unique in that they can appear both in *postfix* form, where they follow the operand as just shown, and *prefix* form, where they precede the operand. In the prefix form, the operand is incremented or decremented before the value is obtained for use in the expression. In postfix form, the previous value is obtained for use in the expression, and then the operand is modified.

3. **The Bitwise Operators**

Java defines several *bitwise operators* which can be applied to the integer types, **long**, **int**, **short**, **char**, and **byte**. These operators act upon the individual bits of their operands.
They are summarized in the following table:

Operator	**Result**
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

The Bitwise Logical Operators The bitwise logical operators are **&**, |, ^, and ~. The following table shows the outcome of each operation. In the discussion that follows, keep in mind that the bitwise operators are applied to each individual bit within each operand.

A	**B**	**A | B**	**A & B**	**A ^ B**	**~A**
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

The Bitwise NOT: Also called the *bitwise complement,* the unary NOT operator, ~, inverts all of the bits of its operand. For example, the number 42, which has the following bit pattern:
00101010 becomes 11010101 after the NOT operator is applied.

The Bitwise AND: The AND operator, **&**, produces a 1 bit if both operands are also 1. A zero is produced in all other cases. Here is an example:

```
00101010        42
&00001111       15
--------------
00001010        10
```

The Bitwise OR; The OR operator, |, combines bits such that if either of the bits in the operands is a 1, then the resultant bit is a 1, as shown here:

```
00101010              42
| 00001111            15
--------------
00101111              47
```

The Bitwise XOR: The XOR operator, ^, combines bits such that if exactly one operand is 1, then the result is 1. Otherwise, the result is zero. The following example shows the effect of the ^. This example also demonstrates a useful attribute of the XOR operation.

Notice how the bit pattern of 42 is inverted wherever the second operand has a 1 bit. Wherever the second operand has a 0 bit, the first operand is unchanged. You will find this property useful when performing some types of bit manipulations.

```
00101010              42
^00001111             15
-------------
00100101              37
```

Using the Bitwise Logical Operators The following program demonstrates the bitwise logical operators:

// A Java Program Demonstrate the bitwise logical operators.

```
public class BitLogic {
public static void main(String args[]) {
String binary[] = {
"0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111",
"1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111"
};
int a = 3; // 0 + 2 + 1 or 0011 in binary
int b = 6; // 4 + 2 + 0 or 0110 in binary
int c = a | b;
int d = a & b;
int e = a ^ b;
int f = (~a & b) | (a & ~b);
int g = ~a & 0x0f;
System.out.println (" a = " + binary[a]);
System.out.println (" b = " + binary[b]);
System.out.println (" a|b = " + binary[c]);
System.out.println(" a&b = " + binary[d]);
System.out.println(" a^b = " + binary[e]);
System.out.println("~a&b|a&~b = " + binary[f]);
System.out.println(" ~a = " + binary[g]);
}
}
```

4. **The unsigned Right Shift Operator:**

In these cases you will generally want to shift a zero into the high-order bit no matter what its initial value was. This is known as an *unsigned shift.* To accomplish this, you will use Java's unsigned, shift-right operator, >>>, which always shifts zeros into the high-order bit.

The following code fragment demonstrates the >>>. Here, **a** is set to -1, which sets all 32 bits to 1 in binary. This value is then shifted right 24 bits, filling the top 24 bits with zeros, ignoring normal sign extension. This sets **a** to 255.

```
int a = -1;
a = a >>> 24;
```

Here is the same operation in binary form to further illustrate what is happening:

```
11111111 11111111 11111111 11111111 -1 in binary as an int
>>>24
00000000 00000000 00000000 11111111 255 in binary as an int
```

The >>> operator is often not as useful as you might like, since it is only meaningful for 32- and 64-bit values.

5. **Relational Operator:**

The *relational operators* determine the relationship that one operand has to the other. Specifically, they determine equality and ordering. The relational operators are shown here:
==, !=, <, >, <=, >=.

The outcome of these operations is a **boolean** value. The relational operators are most frequently used in the expressions that control the **if** statement and the various loop statements.

Any type in Java, including integers, floating-point numbers, characters, and Booleans can be compared using the equality test, ==, and the inequality test, **!=**. Notice that in Java (as in C and C++) equality is denoted with two equal signs, not one. (Remember: a single equal sign is the assignment operator.)

6. **Logical Operators:**

The Boolean logical operators shown here operate only on **boolean** operands. All of the binary logical operators combine two **boolean** values to form a resultant **boolean** value.

Operator	Result	Operator	Result
&	Logical AND	&=	AND Assignment
|	Logical OR	|=	OR assignment
^	Logical XOR (exclusive OR)	^=	XOR assignment
||	Short-circuit OR	==	Equal to
&&	Short-circuit AND	!=	Not Equal to
!	Logical unary NOT		

The logical Boolean operators, **&**, |, and ^, operate on **boolean** values in the same way that they operate on the bits of an integer. The logical **!** operator inverts the Boolean state: **!true == false** and **!false == true**. The following table shows the effect of each logical operation:

A	B	A | B	A & B	A ^ B	!A
False	False	False	False	False	True
True	False	True	False	True	False
False	True	True	False	True	True
True	True	True	True	False	False

Here is a program that is almost the same as the **BitLogic** example shown earlier, but it operates on **boolean** logical values instead of binary bits:

```
// Demonstrate the boolean logical operators.
class BoolLogic {
public static void main(String args[]) {
boolean a = true;
boolean b = false;
boolean c = a | b;
boolean d = a & b;
boolean e = a ^ b;
boolean f = (!a & b) | (a & !b);
boolean g = !a;
System.out.println(" a = " + a);
System.out.println(" b = " + b);
System.out.println(" a|b = " + c);
System.out.println(" a&b = " + d);
System.out.println(" a^b = " + e);
System.out.println("!a&b|a&!b = " + f);
System.out.println(" !a = " + g);
}
}
```

After running this program, you will see that the same logical rules apply to **boolean** values as they did to bits. As you can see from the following output, the string representation of a Java **boolean** value is one of the literal values **true** or **false**:

```
a = true
b = false
a|b = true
a&b = false
a^b = true
a&b|a&!b = true
!a = false
```

Difference between && and &

While working with ''&&'', the second condition will evaluated provided that the 1st is satisfied. Whereas in '&' the 2nd condition will be evaluated irrespective of whether 1st one is successful or not i.e.2nd condition is compulsorily checked out.

Difference between || and |

In ''||' the second condition will be evaluated provided that the 1st one is failed. In case if the 1st one is returning true, second condition will not be evaluated. If we want to compulsorily evaluate the 2nd condition also we can use''|' (or).

7. **The Conditional Operator**

Java includes a special *ternary* (three-way) *operator* that can replace certain types of if then-else statements. This operator is the **?**, and it works in Java much like it does in C and C++. It can seem somewhat confusing at first, but the **?** can be used very effectively once mastered. The **?** has this general form:

expression1 ?expression2 : expression3

Here, *expression1* can be any expression that evaluates to a **boolean** value. If *expression1* is **true**, then *expression2* is evaluated; otherwise, *expression3* is evaluated. The result of the **?** operation is that of the expression evaluated. Both *expression2* and *expression3* are required to return the same type, which can't be **void**. Here is an example of the way that the ? is employed:

ratio = denom == 0 ? 0 : num / denom;

8. **Assignment Operator: =**

Control Statements

The statements that make interpreter/compiler execute the statements of the program depending on a condition. Thus they control the flow of control. Three kinds of control statement are known:

1. Selection Control Construct.
2. Iteration Construct.
3. Jump Statement.

Selection Control Construct: These statements of the programming language enable the programmer to put up a condition and then take a decision. The result of the condition determines the statement to be executed. This function is performed by **if** and **switch** statement.

General format of if statement:

1. **if**

```
if (condition or Boolean variable)
statement;
```

2. **blocked if**

```
if (condition or Boolean variable)
    {
            Statement1;
            Statement2;
            Statement3;
             ……….
    }
```

3. **if…else**

```
if (condition or Boolean variable)
        statement;
else
        statement;
```

4. **nested if**

```
if (condition or Boolean variable){
if (condition or Boolean variable)
statement;
}
```

5. **nested if…else**

```
if (condition or Boolean variable)
{
if (condition or Boolean variable)
statement;
else
statement;
}
```

General format of switch…. Case

```
Switch (expression)
        {
                case <value>:
                ……………
                ……………
                break;
                case <value>:
                ……………
                ……………
                break;
                default:
                ………..
        }
```

NOTE: Expression must should generate value i.e. byte, char, int, short (any one of four only).

Type Casting: While evaluating an expression. If the variables are of dissimilar types, all the variables will be automatically type casted into a high ranged data type available in the expression. The resultant value will be of the high range data type itself. In case if we want it in any other data type format, explicitly type cast the result.

```
short a=10;
byte b=20;
int  c=30;
long d=1000;
```

a+b+c+d; The resulted value is in long data type because java compiler type casted internally into higher ranged data type.

Note-

E.g. int I=10, int j=20;

```
1. if (I >20 && I< j++)
{
……………
}
Output:  I=10 ; j=20

2.if (I>5 && I< j++)
{
……………
}
Output:  I=10  j=21

3. if (I>20 & I<j++)
{
}
Output: I=10  I=21
```

New Feature of JAVA SE 6

1. **Changes in I/O**

This is a new feature added in Java SE 6, which has the ability to read text from a terminal without having it echo on the screen through java.io.Console.

2. **Collections Framework Enhancement**

In Collection framework, we are able to improve the performance hashing function that is used by java.util.HashMap. It provides some new Collection interfaces also.

3. **Changes in jar and zip**

Jar and zip support has been enhanced in JDK 6. In this, two new compressed streams have been added. These are java.util.zip.DeflaterInputStream and java.util.zip.InflaterOutputStream.

4. **Java Web Start enhancements in version 6**

The cache format of Java Web Start and Plug-In has been fully changed. All dialogs have been redesigned and consolidated between Java Web Start and Plug-In to be more user friendly.

5. **JMX API Enhancements**

Enhancements has been made to the JMX API in Java SE 6 like : JMX API now completely supports Generics, MXBeans have been added and much more…

6. **Java Platform Debugger Architecture Enhancements**

JVMDI has been deleted in JAVA SE 6 and JPDA has been added. Some new Methods are included like boolean boolean canGetInstanceInfo() and much more…

7. **Java SE 6 Monitoring and Management Enhancements**

API provided by JAVA SE allows you to monitor and manage the JVM in the package java.lang.management. JAVA SE 6 includes some enhancement to this API.

8. **New Package java.util.spi in JDK 6**

A new package Java.util.spi has been included in JAVA SE 6.

9. **Networking features and enhancements in Java SE version 6.0**

This feature include enhancement in networkInterface, support for Internationalized Domain Names, HTTP Negotiate Authentication and much more…

10. **Enhancements in java.lang.Class and java.lang.reflect**

Some new Methods are included in java.lang.Class like : getInterfaces(), getClasses(), getConsturctors(), getMethod(String, Class…) and much more….

11. **Enhancement in RMI for JDKTM 6**

java.rmi.MarshalledObject now support generics

12. **JAVA SE 6 Security Enhancements**
JAVA SE 6 has added support for some security functionality: the XML Digital signature API and implementation, Smart Card I/O API. And much more…

13. **Serialization Changes and Enhancements in JAVA SE Development Kit 6**
The new method ObjectStreamClass.lookupAny now used to obtain an ObjectStreamClass instance for a non-serializable Class

14. **JavaTM Virtual Machine Technology**
DTrace support has include in Java SE 6 HotSpot VM. The hotspot provider makes available probes that can be used to track the lifespan of the VM, thread start and stop events

15. **Scripting for the Java Platform**
By using this Features developers integrate Java technology and scripting languages by defining a standard framework and application programming interface (API)

16. **Leveraging Security in the Native Platform Using Java SE 6 Technology**
The JAVA SE 6 provide a large set of security APIs. These Security APIs span a wide range of areas, including cryptography public key infrastructure, secure communication, authentication, and access control.

17. **JAX-Web Services 2.0 With the Java SE 6 Platform**
Most exciting new features of the JAVA SE 6 is support for the Java API for XML Web Services (JAX-WS), version 2.0.