# Introduction to JAVA

## 1.0 Introduction:

Java is a high-level, third generation programming language, like C, FORTRAN, Smalltalk, Perl, and many others. You can use Java to write computer applications that play games, store data or do any of the thousands of other things computer software can do. Compared to other programming languages, Java is most similar to C. However although Java shares much of C's syntax, it is not C. Knowing how to program in C or, better yet, C++, will certainly help you to learn Java more quickly, but you don't need to know C to learn Java. A Java compiler won't compile C code, and most large C programs need to be changed substantially before they can become Java programs. What's most special about Java in relation to other programming languages is that it lets you write special programs called applets that can be downloaded from the Internet and played safely within a web browser. Java language is called as an Object-Oriented Programming language and before begining for Java, we have to learn the concept of OOPs(Object-Oriented Programming).

Java is a general-purpose, object-oriented programming language developed by Sun Microsystems of USA in 1991.Originally called Oak by James Gosling (one of the inventor of the language). Java was invented for the development of software for consumer electronic devices like TVs, Washing- machine, microwave oven etc. The main aim had to make java simple, portable and reliable.

## 1.1 Basic Concept of OOP(Object-Oriented Programming):

There are some basic concepts of object oriented programming as follows:

1. Object
2. Class
3. Data abstraction
4. Data encapsulation
5. Inheritance
6. Polymorphism
7. Dynamic binding

1. **Object** Objects are important runtime entities in object oriented method. They may characterize a location, a bank account, and a table of data or any entry that the program must handle.

For example:

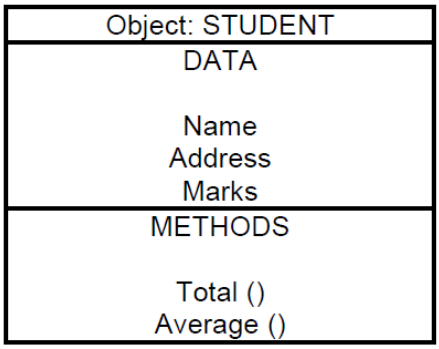



Fig.1.1 Representation of an object ―STUDENT‖

Each object holds data and code to operate the data. Object can interact without having to identify the details of each other's data or code. It is sufficient to identify the type of message received and the type of reply returned by the objects.

Another example of object is CAR

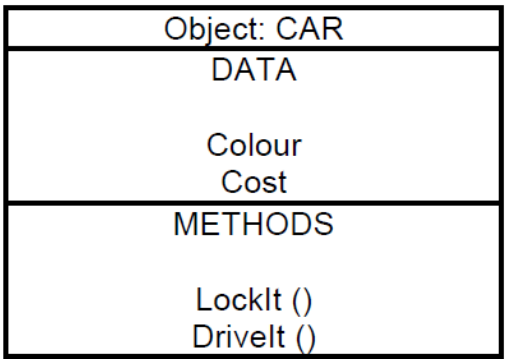



Fig.1.1 and Fig.1.2 shows actual representation of object.

**2. Classes**

A class is a set of objects with similar properties (attributes), common behaviour (operations), and common link to other objects. The complete set of data and code of an object can be made a user defined data type with the help of class.

The objects are variable of type class. A class is a collection of objects of similar type. Classes are user defined data types and work like the build in type of the programming language. Once the class has been defined, we can make any number of objects belonging to that class. Each object is related with the data of type class with which they are formed.
As we learned that, the classification of objects into various classes is based on its properties (States) and behavior (methods). Classes are used to distinguish are type of object from another. The important thing about the class is to identify the properties and procedures and applicability to its instances.

**For example:** Vehicle

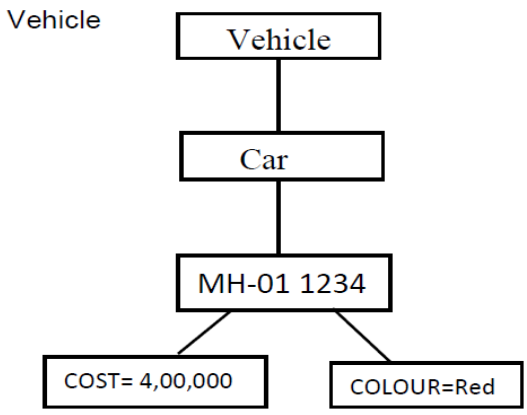



**Fig.1.3 Representation of class**

In above example, we will create an objects MH-01 1234 belonging to the class car. The objects develop their distinctiveness from the difference in their attribute value and relationships to other objects.

**3. Data Abstraction**

Data abstraction refers to the act of representing important description without including the background details or ---explanations.

Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, cost and functions operate on these attributes. They summarize all the important properties of the objects that are to be created.

Classes use the concepts of data abstraction and it is called as Abstract Data Type (ADT).

**4. Data Encapsulation**

Data Encapsulation means wrapping of data and functions into a single unit (i.e. class). It is most useful feature of class. The data is not easy to get to the outside world and only those functions which are enclosed in the class can access it. These functions provide the boundary between Object's data and program. This insulation of data from direct access by the program is called as Data hiding. For example:

5. **Inheritance**

Inheritance is the process by which objects of one class can get the properties of objects of another class. Inheritance means one class of objects inherits the data and behaviours from another class. Inheritance maintains the hierarchical classification in which a class inherits from its parents.

Inheritance provides the important feature of OOP that is reusability. That means we can include additional characteristics to an existing class without modification. This is possible deriving a new class from existing one.

In other words, it is property of object-oriented systems that allow objects to be built from other objects. Inheritance allows openly taking help of the commonality of objects when constructing

new classes. Inheritance is a relationship between classes where one class is the parent class of another (derived) class. The derived class holds the properties and behaviour of base class in addition to the properties and behaviour of derived class.

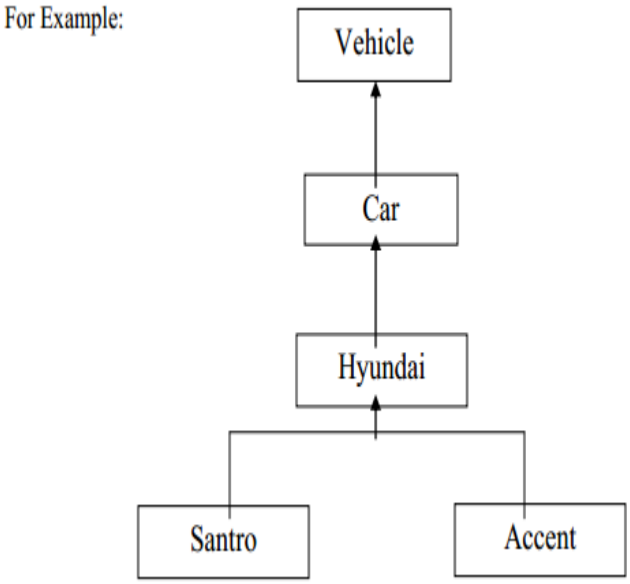



**Fig.1.5 Inheritance**

In Fig.1.5, the Santro is a part of the class Hyundai which is again part of the class car and car is the part of the class vehicle. That means vehicle class is the parent class.

**6. Polymorphism**

(Poly means "many" and morph means "form"). Polymorphism means the ability to take more than one form. Polymorphism plays a main role in allocate objects having different internal structures to share the same external interface. This means that a general class of operations may be accessed in the same manner even though specific activities associated with each operation may differ. Polymorphism is broadly used in implementing inheritance.
It means objects that can take on or assume many different forms. Polymorphism means that the same operations may behave differently on different classes. ***Booch*** *defines polymorphism as the relationship of objects many different classes by some common super class.* Polymorphism allows us to write generic, reusable code more easily, because we can specify general instructions and delegate the implementation detail to the objects involved.

**For Example:**

In a pay roll system, manager, office staff and production worker objects all will respond to the compute payroll message, but the real operations performed are object particular.

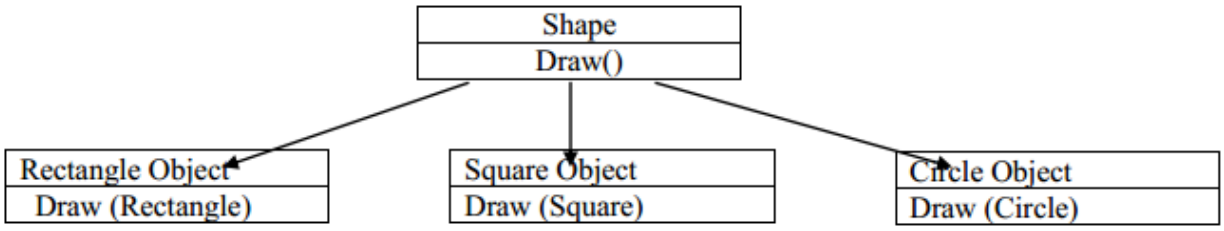



Fig.1.6 Polymorphism

**7. Dynamic Binding**

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code related with a given procedure call is not known until the time of the call at run time. Dynamic binding is associated polymorphism and inheritance.

**1.2 Java History:**

James Gosling 

is a famous Canadian software developer who has been with Sun Microsystems since 1984 and is considered as father of Java programming language. Gosling did the original design of Java and implemented its original compiler and virtual machine.

- In1990, Sun Micro Systems Inc. (US) was conceived a project to develop software for consumer electronic devices that could be controlled by a remote. This project was called Stealth Project but later its name was changed to Green Project.
- In January 1991, Project Manager James Gosling and his team members Patrick Naughton, Mike Sheridan, Chris Wrath, and Ed Frank met to discuss about this project.
- Gosling thought C and C++ would be used to develop the project. But the problem he faced with them is that they were system dependent languages. The trouble with C and C++ (and most other languages) is that they are designed to be compiled for a specific target and could not be used on various processors, which the electronic devices might use.
- James Gosling with his team started developing a new language, which was completely system independent. This language was initially called OAK. Since this name was registered by some other company, later it was changed to Java.
- James Gosling and his team members were consuming a lot of coffee while developing this language. Good quality of coffee was supplied from a place called "Java Island'. Hence they fixed the name of the language as Java. The symbol for Java language is cup and saucer.
- Sun formally announced Java at Sun World conference in 1995. On January 23rd 1996, JDK1.0 version was released. Bill Joy, Arthur van Hoff, Jonathan Payne, Frank Yellin, and Tim Lindholm were key contributors to the maturing of the original prototype.
- the similarities between Java and C++, it is tempting to think of Java as simply the "Internet version of C++." However, to do so would be a large mistake. Java has significant practical and philosophical differences.
- While it is true that Java was influenced by C++, it is not an enhanced version of C++. For example, Java is neither upwardly nor downwardly compatible with C++. Of course, the similarities with C++ are significant, and if you are a C++ programmer, then you will feel right at home with Java.
- One other point: Java was not designed to replace C++. Java was designed to solve a certain set of problems. C++ was designed to solve a different set of problems. Both will coexist for many years to come.

**1.3 JAVA FEATURES:**

**Features of Java**

- **Simple**: Learning and practicing java is easy because of resemblance with c and C++.
- **Object Oriented Programming Language**: Unlike C++, Java is purely OOP.
- **Distributed**: Java is designed for use on network; it has an extensive library which works in agreement with TCP/IP.

- **Secure:** Java is designed for use on Internet. Java enables the construction of virus-free, tamper free systems. Using Java Compatible Browser, anyone can safely download Java applets without the fear of viral infection or malicious intent. **Java achieves this protection by confining a Java program to the Java execution environment and by making it inaccessible to other parts of the computer**. We can download applets with confidence that no harm will be done and no security will be breached.

- **Robust (Strong/ Powerful):** Most programs in use today fail for one of the two reasons: memory management or exceptional conditions. Thus, the ability to create robust programs was given a high priority in the design of Java. Java forces the user to find mistake in the early stages of program development. Java checks code at compilation time. However, it also checks the code at run time also. **Java programs will not crash because of its exception handling and its memory management features**.

- **Interpreted:** Java programs are compiled to generate the byte code. This byte code can be downloaded and interpreted by the interpreter. .class file will have byte code instructions and JVM which contains an interpreter will execute the byte code.

- **Portable**: In Java, many types of computers and operating system are in use throughout the world and are connected to the Internet. For downloading programs through different plateforms connected to the Internet, some portable, executable code is needed. **Java does not have implementation dependent aspects and it yields or gives same result on any machine.**

- **Architectural Neutral Language:** Java development team work on the philosophy of **write once; run anywhere, anytime, forever**" and as a result the Java Virtual Machine (JVM) was developed. Only the JVM can execute the bytecode. Java byte code is not machine dependent, it can run on any machine with any processor and with any OS.

- **High Performance**: Along with interpreter there will be JIT (Just In Time) compiler which enhances the speed of execution.

- **Multithreaded**: Executing different parts of program simultaneously is called multithreading. This is an essential feature to design server side programs.

- **Dynamic**: We can develop programs in Java which dynamically change on Internet (e.g.: Applets).

## 1.4 COMPARISON IN JAVA AND C++

| | |
|---|---|
| 1. Java is true Objectoriented language. | C++ is basically C with Object-oriented extension. |
| 2. Java does not support operator overloading. | C++ supports operator overloading. |
| 3. It supports labels with loops and statement blocks | It supports goto statement. |

| | |
|---|---|
| 4. Java does not have template classes as in C++. | C++ has template classes. |
| 5. Java compiled into byte code for the Java Virtual Machine. The source code is independent on operating system. | Source code can be written to be platform independent C++ typically compiled into machine code. |
| 6. Java does not support multiple inheritance of classes but it supports interface. | C++ supports multiple inheritance of classes. |
| 7. Runs in a protectedvirtual machine. | Exposes low-level system facilities |
| 8 Java does not support global variable. Every variable should declare in class. | C++ support global variable. |
| 9 Java does not use pointer | C++ uses pointer. |
| 10. It Strictly enforces an object oriented Programming paradigm. | It Allows both procedural programming and object oriented programming. |
| 11 There are no header files in Java. | We have to use header file in C++. |

## 1.5 JAVA ENVIRONMENT

Java environment includes a large number of development tools and hundreds of classes and methods. The development tools are part of the system known as ***Java Development Kit*** (JDK) and classes and methods are part of the ***Java standard Library*** (JSL), also known as the ***Application Programming Interface*** (API).

### 1.5.1 JAVA DEVELOPMENT KIT

The Java Development Kit comes with a collection of tools that are used for developing and running Java programs. They include:

**Appletviewer (for running Java applets)** : Enable us to run Java Applets(without actually using a Java-compatible browser).

**Javac (Java compiler)** : Which translates Java sourcecode to bytecode files that interpreter can understand.

**Java (Java Interpreter)** : Which runs applets and applications by reading and interpreting bytecode.

**Javap(Java disassemble)** : Which enables us to convert bytecode files into a program description.

**Javah (for C header files)** : Produce header file for use with native methods.

**Javadoc (for creating HTML documents)** : Create HTML-format documentation from Java sourcecode files.

**Jdb(Java debugger)** : Which helps use to find errors in our programs.

### 1.5.2 APPLICATION PROGRAMMING INTERFACE

The Java Standard Library (or API) includes hundreds of classes and methods grouped into several functional packages. Most commonly used packages are:

**Language support Package:** A collection of classes and methods required for implementing basic features of Java.

**Utilities packages:** A collection of classes to provide utility functions such as date and time functions.

**Input/Output package:** A collection of classes required for input/output manipulation.

**Networking package:** A collection of classes for communicating with other computers via internet.

**Applet package:** This includes as set of classes that allows us to create Java applets.

### 1.5.3 JAVA RUNTIME ENVIRONMENT

The Java runtime Environment (JRE) allows the execution of programs developed in Java. It primarily comprises of the following:

**Java Virtual Machine:** It converts bytecode into machine code.

**Runtime class libraries:** These are the set of core class libraries that are required for the execution of Java program

User Interface toolkits: AWT and Swings are the examples of toolkit the support varied input methods for the users to interact with the application program.

## 1.6 JAVA VIRTUAL MACHINE

Java Virtual Machine (JVM) is the heart of entire Java program execution process. First of all, the .java program is converted into a .class file consisting of byte code instructions by the java compiler at the time of compilation. Remember, this java compiler is outside the JVM. This .class file is given to the JVM.

Following figure shows the architecture of Java Virtual Machine.

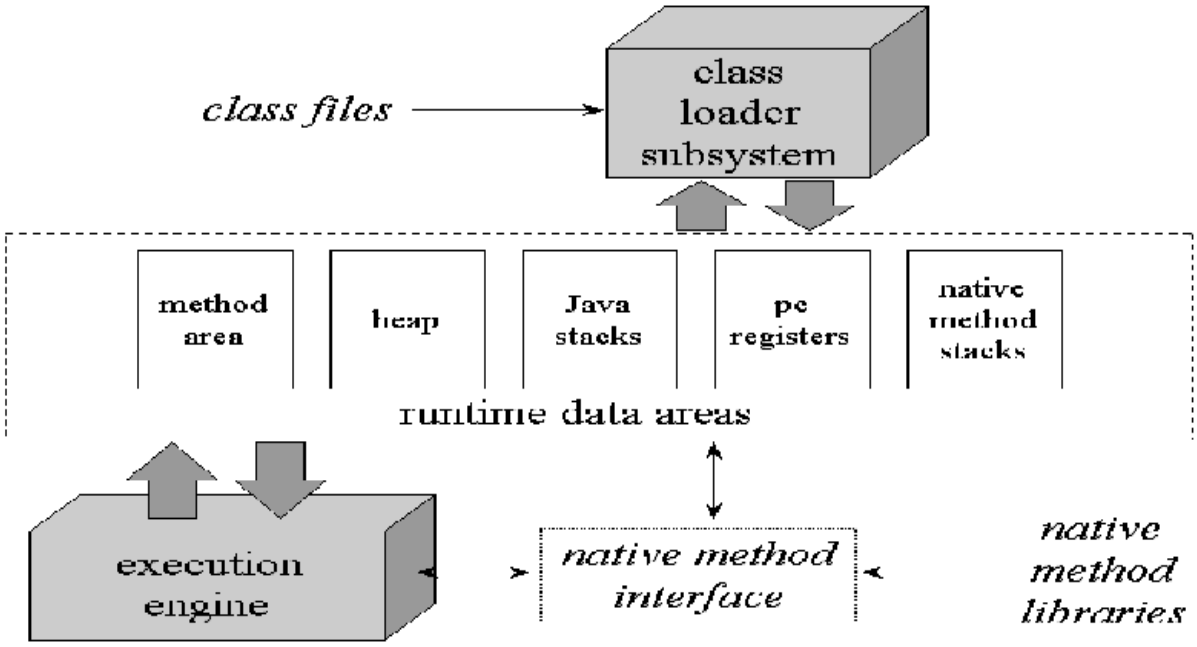



**Figure 2.3:** The internal architecture of the Java virtual machine.

In JVM, there is a module (or program) called class loader sub system, which performs the following instructions:

- First of all, it loads the .class file into memory.
- Then it verifies whether all byte code instructions are proper or not. If it finds any instruction suspicious, the execution is rejected immediately.
- If the byte instructions are proper, then it allocates necessary memory to execute the program. This memory is divided into 5 parts, called ***run time data areas***, which contain the data and results while running the program. These areas are as follows:
- **Method area:** Method area is the logical memory block of JVM, which holds information about classes and interfaces. Static variables are treated as class variables, because they take memory from method area. The size of the *method area* shrinks and expands according to the size of the application.
- **Heap:** In Java when an object or an array is created, memory is allocated to them from heap. The JVM through the use of new operator allocates memory from the heap for an object. The JVM has a daemon thread known as Garbage Collector whose task is to free those object from heap whose reference is not alive in stack.
- **Java Stacks:** Method code are stored on Method area. But while running a method, it needs some more memory to store the data and results. This memory is allotted on Java Stacks. So, Java Stacks are memory area where Java methods are executed.
- **PC (Program Counter) registers:** It keep track of the sequence of execution of the program. These are the registers (memory areas), which contain memory address of the instructions of the methods or PC register or program counter register holds the addresses of the instructions to be executed next.

- **Native Method Stacks:** When a Java application invokes a native method, that application does not only use *Java Stacks* but also uses the *native method stack* for the execution of native methods (for example C/C++ functions). To execute the native methods, generally native method libraries (for example C/C++ header files) are required. The libraries required for the execution of native methods are available to the Java Virtual Machine through **Java Native Interface**.
- **Execution Engine** contains interpreter and JIT compiler which translates the byte code instructions into machine language which are executed by the microprocessor. Hot spot (loops/iterations) is the area in .class file i.e. executed by JIT compiler. JVM will identify the Hot spots in the .class files and it will give it to JIT compiler where the normal instructions and statements of Java program are executed by the Java interpreter.

**Byte Code:** It is the unique characteristic property of the Java programming language. It is something like a normal text file. Therefore, it cannot be affected by virus. It is an intermediate between a human readable source and machine readable source. Byte codes are plateform-independent. Therefore, JVM is plateform dependent to make Java programming plateform indepdent.

### 1.7 Process of Compiling and Running a Java Application Program

All Java source code is written in text editor and saved with the .Java extension. Source code file are compiled into .class file by the java compiler. A .class file contains byte codes (plate-form independent intermediate code between a human readable source and machine readable source).

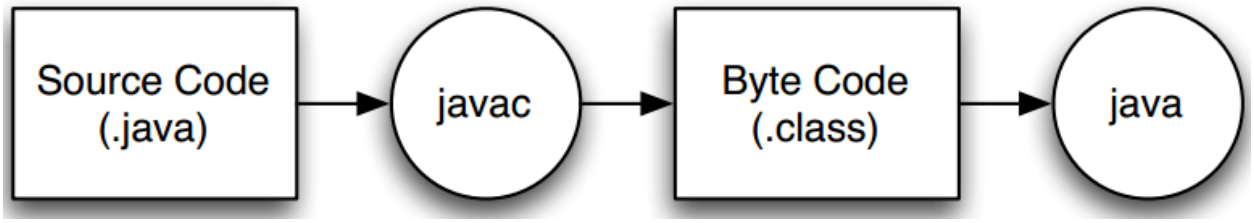



Figure 2.1 Abstract view of Java program execution.

JVM works on different operating systems. The .class files (bytecode) can run on various operating systems. JVM converts bytecode into computer-readable machine code that can be executed.

**Compiling:** is the process of translating source code written in a particular programming language into computer-readable machine code that can be executed.

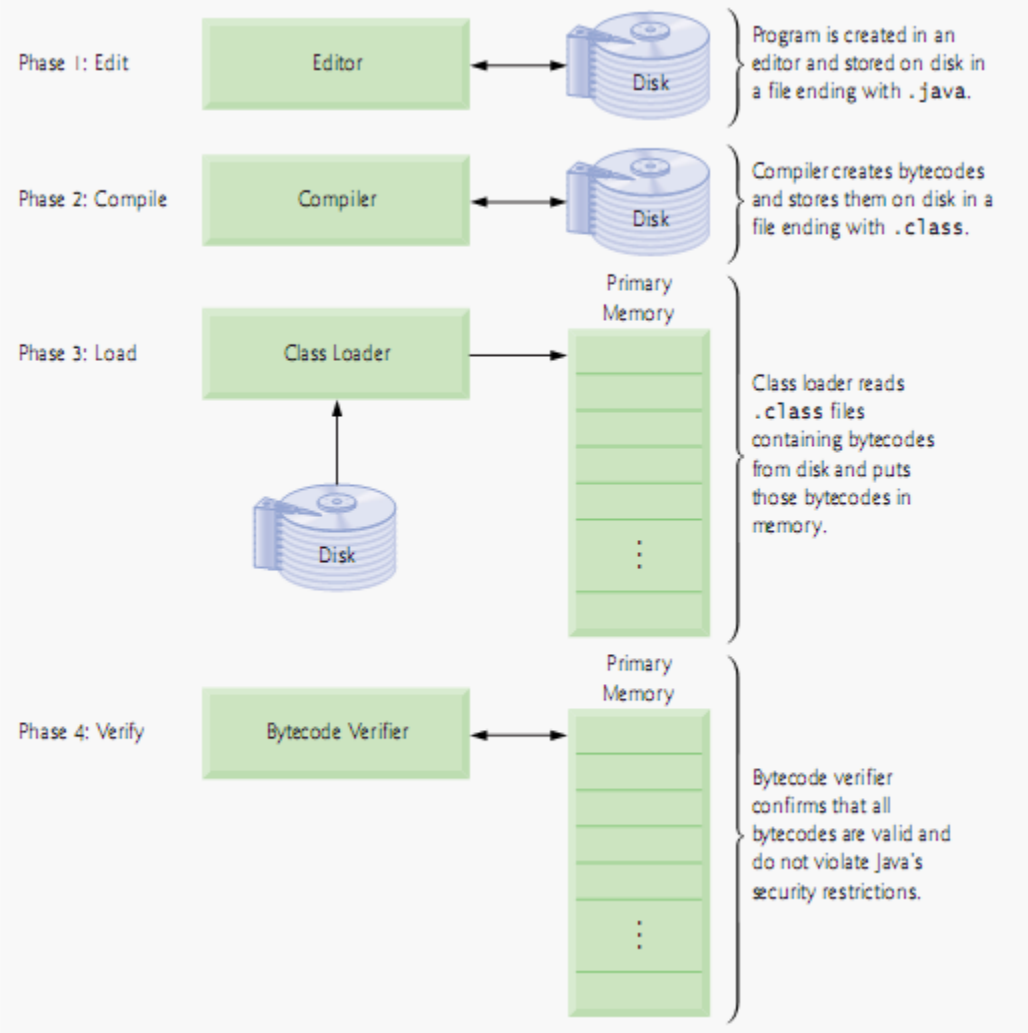
## C:\> javac Sample.java

This command will produce a file 'Sample.class', which is used for running the program with the command 'java'.

Running: is the process of executing program on a computer.

**C:\> java Sample**

## Steps Of Java Program Execution



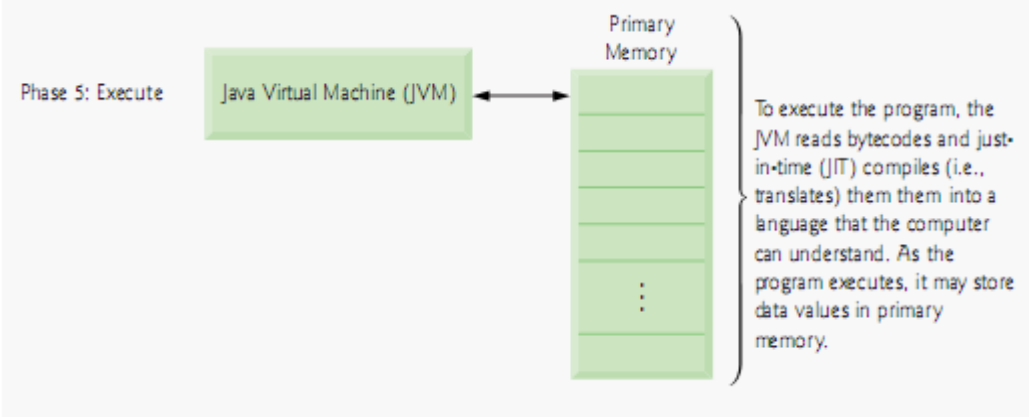



Figure 2.2 shows the entire steps of Java program execution

## Editions of Java Technology

**Table 2–1 Java Jargon**

| Name | Acronym | Explanation |
|---|---|---|
| Java Development Kit | JDK | The software for programmers who want to write Java programs |
| Java Runtime Environment | JRE | The software for consumers who want to run Java programs |
| Standard Edition | SE | The Java platform for use on desktops and simple server applications |
| Enterprise Edition | EE | The Java platform for complex server applications |
| Micro Edition | ME | The Java platform for use on cell phones and other small devices |
| Java 2 | J2 | An outdated term that described Java versions from 1998 until 2006 |
| Software Development Kit | SDK | An outdated term that described the JDK from 1998 until 2006 |
| Update | u | Sun's term for a bug fix release |
| NetBeans | — | Sun's integrated development environment |

## Jdk Directory Structure

**Table 2–2 Java Directory Tree**

| Directory Structure | Description |
|---|---|
| *jdk* | (The name may be different, for example, jdk1.8) |
| bin | The compiler and tools |
| demo | Look here for demos |
| docs | Library documentation in HTML format (after expansion of j2sdk*version*-doc.zip) |
| include | Files for compiling native methods (see Volume II) |
| jre | Java runtime environment files |
| lib | Library files |
| src | The library source (after expanding src.zip) |

**Obtaining the Java Environment (JDK) :**

- Install JDK after downloading, by default JDK will be installed in

C:\Program Files\Java\jdk1.7.0_21 (Here jdk1.7.0_21 is JDK's version)

Setting up Java Environment: After installing the JDK, we need to set at least one environment variable in order to able to compile and run Java programs. A PATH environment variable enables the operating system to find the JDK executables when our working directory is not the JDK's binary directory.

- Setting environment variables from a command prompt: If we set the variables from a command prompt, they will only hold for that session. To set the PATH from a command prompt: set PATH=C:\Program Files\Java\jdk1.5.0_05\bin;.%PATH%
- Setting environment variables as system variables: If we set the variables as system variables they will hold continuously.

o Right-click on My Computer

o Choose Properties

o Select the Advanced System Settings. (Under control panel home left pane)

o Click the Environment Variables button at the bottom.

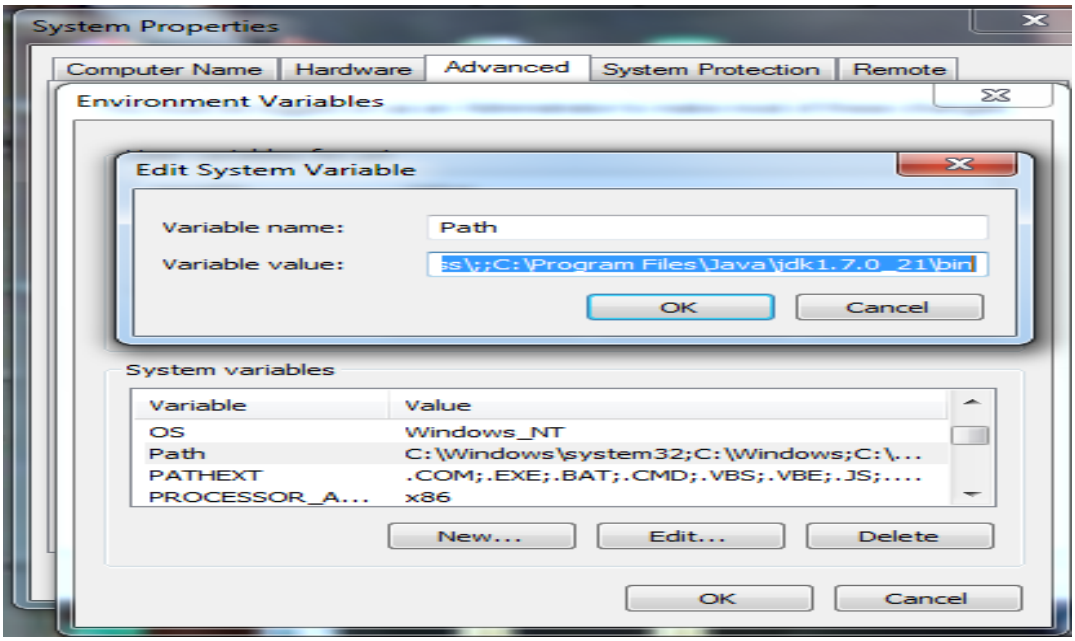
o In **system variables** tab, select path (system variable) and click on edit button

o A window with variable name- path and its value will be displayed.

o Don't disturb the default path value that is appearing and just append (add) to that path at the end:;C:\ProgramFiles\Java\ jdk1.7.0_21\bin;.;

o Finally press OK button.

- Repeat the process and type at:
  **Variable name class path**
  Variable value:C:\ProgramFiles\Java\ jdk1.7.0_21\lib\tool.jar;.;
- Finally press OK button.




**Structure of the Java Program:**

As all other programming languages, Java also has a structure.

- There could be only one public class per source code file but it can have multiple non public class.
- In case there is any public class present in source code file, name of the file should be the class name.
- Sequence of difference statement in source code file would be ***package*** >> ***import*** >> ***Class declaration.***
- No Sequence rule is applied for *Comments. Comments* can be there in any part of the source code file at any location.
- File with no public class can have any name there is no rule applied for the same.
- ***Import*** and ***package*** statements applied to all the classes in same source code file.

## Rules Applied on Java Source Code File

Classes are written in Java source file. A source file can contain more than one Java class. Below are the rules related to Java source code file?

- The first line of the C/C++ program contains include statement. For example, <stdio.h> is the header file that contains functions, like printf (), scanf () etc. So if we want to use any of these functions, we should include this header file in C/ C++ program.

- Similarly in Java first we need to import the required packages. By default **java.lang.*** is imported. Java has several such packages in its library.
- A package is a kind of directory that contains a group of related classes and interfaces. A class or interface contains methods.
- Since Java is purely an Object Oriented Programming language, we cannot write a Java program without having at least one class or object.
- So, it is mandatory to write a class in Java program. We should use class keyword for this purpose and then write class name.
- In C/C++, program starts executing from main method similarly in Java, program starts executing from main method. The return type of main method is void because program starts executing from main method and it returns nothing.

## Naming Conventions

Naming conventions specify the rules to be followed by a Java programmer while writing the names of packages, classes, methods etc.

- Package names are written in small letters.

  **e.g.: java.io, java.lang, java.awt etc**

- Each word of class name and interface name starts with a capital

  **e.g.: Sample, AddTwoNumbers**

- Method names start with small letters then each word start with a capital

  **e.g.: sum (), sumTwoNumbers (), minValue ()**

- Variable names also follow the same above method rule

  **e.g.: sum, count, totalCount**

- Constants should be written using all capital letters

  **e.g.: PI, COUNT**

- Keywords are reserved words and are written in small letters.

  **e.g.: int, short, float, public, void**

## First Java Program

**Sample Program:**

```
//A Simple Java Program
import java.lang.System;
import java.lang.String;
class Sample
{
        public static void main(String args[])
                {
                        System.out.print ("Hello world");
                }
}
```

- Since Java is purely an Object Oriented Programming language, without creating an object to a class it is not possible to access methods and members of a class.
- But main method is also a method inside a class, since program execution starts from main method we need to call main method without creating an object.
- Static methods are the methods, which can be called and executed without creating objects.
- Since we want to call main() method without using an object, we should declare main() method as static. JVM calls main() method using its Classname.main() at the time of running the program.
- JVM is a program written by Java Soft people (Java development team) and main() is the method written by us. Since, main () method should be available to the JVM, it should be declared as public. If we don't declare main () method as public, then it doesn't make itself available to JVM and JVM cannot execute it.
- JVM always looks for main() method with String type array as parameter otherwise JVM cannot recognize the main() method, so we must provide String type array as parameter to main () method. Main() method having signature with default variable arguments.

- A class code starts with a {and ends with a}.
- A class or an object contains variables and methods (functions). We can create any number of variables and methods inside the class.
- This is our first program, so we had written only one method called main().
- Our aim of writing this program is just to display a string "Hello world".
- In Java, print() or write() method is used to display something on the monitor.
- A method should be called by using **objectname.methodname()**. So, to call **print()** method, create an object to **PrintStream** class then call **objectname.print()** method.
- An alternative is given to create an object to **PrintStream Class** i.e. **System.out**.
- Here, **System** is the class name and **out** is a static variable in System class **out** is called a field in System class.
- When we call this field a **PrintStream class object will be created internally**. So, we can call print() method as: System.out.print ("Hello world");
- println() is also a method belonging to PrintStream class. It throws the cursor to the next line after displaying the result.
- In the above Sample program System and String are the classes present in java.lang package.