Unit 1.1
Programming in Java

Introduction

- Java is general purpose, object oriented, high level programming language which can be used to develop different applications like desktop application, network application, web application, mobile application, distributed application etc.

Characteristics of Java

1. Simple: Java is designed to be easy to learn. If we understand the basic concept of OOP, java would be easy to master.
2. Object oriented: Java is a pure object oriented programming language because everything in java is an object and everything in java is defined within a class.
3. Secured: Java is secured programming which enable us to develop virus free network based/distributed application.
4. Platform Independent and portability: Java compiler generates a platform independent (architectural neutral) intermediate code or object code called bytecode (.class file) which can be executed on any processor. The philosophy of the java is "Write once, Run everywhere anytime forever".
5. Compiled and interpreted: Java is a two stage system because it is both compiled and interpreted. In first stage, java compiler translates the source code(.java) into bytecodes(.class). In second phase, java interpreter generates **native** machine code that can be directly executed by the machine that is running the java program.
6. Distributed: Java is designed as a distributed language for creating application on network.
7. Multi-threaded: With java's multithreaded feature, it is possible to write a program that can do many task simultaneously.
8. High performance: The performance of java program is high mainly due to the use of intermediate byte code and multithreading concept.

Java Architecture: Java Virtual Machine and Java Byte code

- In java programming language, all source code is first written in plain text files ending with .java extension.
- Those source code are then compiled into .class files using **javac** compiler.
- .class files are the architectural neutral intermediate code called bytecode which can be run using a special program that act as machine called JVM (Java Virtual Machine). JVM is a run time system for java.

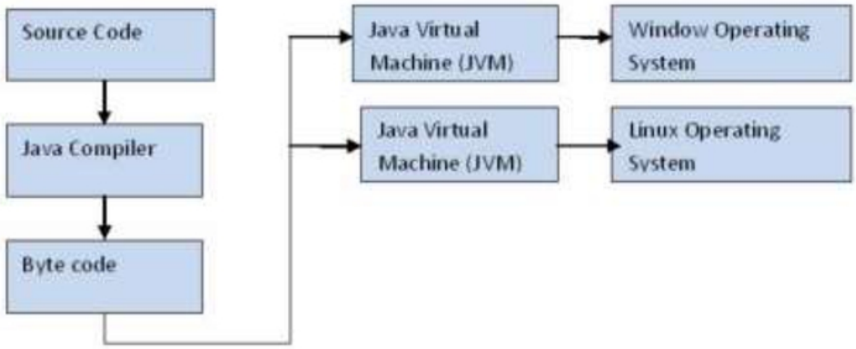


Fig: Java Architecture

- Byte code are not machine dependent code, so to generate the machine code, JVM interprets the bytecode.
- Since JVM is available on different operating system, the same bytecode is capable of running on system.

JRE

- JRE is an acronym for Java Runtime Environment.
- The Java Runtime Environment is a set of software tools which are used for developing and running Java applications.

JRE=JVM+ Library Classes

- It is used to provide the runtime environment.
- It is the implementation of JVM.
- It physically exists. It contains a set of libraries and other files that JVM uses at runtime.

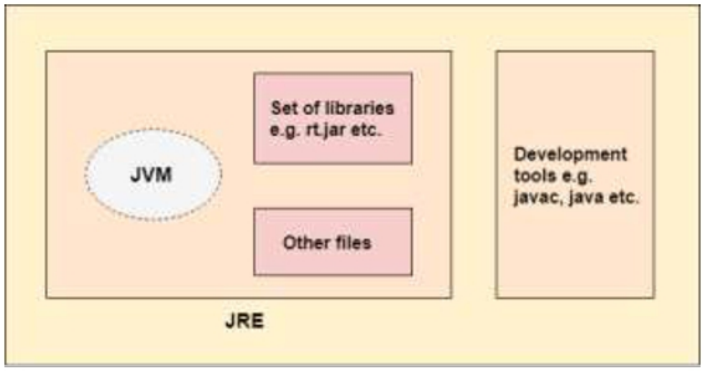
JRE

JDK

- JDK is an acronym for Java Development Kit.
- The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets.

JDK=JRE + Development Tools

- It physically exists.
- It contains JRE + development tools.
- JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:
  - Standard Edition Java Platform: Desktop Based Application, Applet
  - Enterprise Edition Java Platform: Web Based Application
- The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), etc. to complete the development of a Java Application.



JDK

JDK VS JVM VS JRE

- JDK – Java Development Kit (in short JDK) is Kit which provides the environment to develop and execute(run) the Java program. JDK is a kit (or package) which includes two things
  i. Development Tools (to provide an environment to develop your java programs)
  ii. JRE (to execute your java program).

JDK is only used by Java Developers.

- JRE – Java Runtime Environment (to say JRE) is an installation package which provides environment to only run (not develop) the java program (or application) onto your machine.
- JRE is only used by them who only wants to run the Java Programs i.e. end users of your system.

- JVM – Java Virtual machine(JVM) is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for executing the java program line by line hence it is also known as interpreter.

Steps for Compiling and Running Java Programs: Program to display "Hello World"

1. Open any text editor like notepad.
2. Write the source code in java.

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

3. Save the source code with file name exactly same as that of class and extension .java i.e. Demo.java
4. Open command prompt and navigate to folder where we saved the source file.
5. Then compile the source file using java compiler **Javac** using the command as below

   javac Demo.java
6. If no error occurred, then intermediate code called as byte code with extension .class is created i.e. a file Demo.class will be created.
7. Finally, we run the obtained class file using java interpreter tool **Java** using the command as below

   java Demo
8. We will see "Hello World" as output.

Note:

- System.out.println is a Java statement that prints the argument passed, into the System.out which is generally stdout.
  System is a Class
  out is a Variable
  println() is a method
- System is a class in the java.lang package .
- The out is a static member of the System class, and is an instance of java.io.PrintStream .
- The println is a method of java.io.PrintStream. This method is overloaded to print message to output destination, which is typically a console or file.
- The System class belongs to java.lang package

Path and Classpath

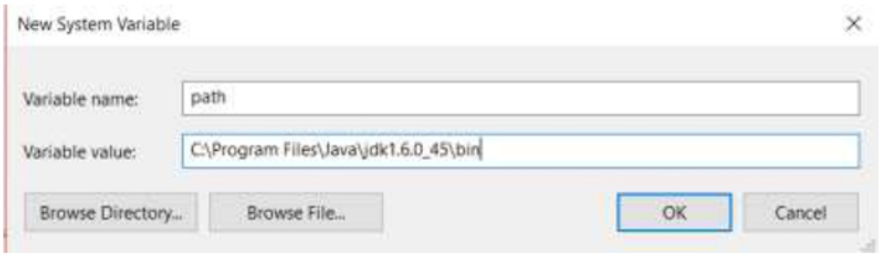
- Environment variables are global system variables accessible by all the processes/users running under the Operating System (OS), such as Windows, macOS and Linux. Environment variables are useful to store system-wide values, for examples, Path and Classpath
- The Path points to the location of the JRE i.e. the java binary files such as the jvm, JDK and necessary libraries.
- The Classpath points to the classes you developed so that the JVM can find them and load them when you run your project.

For Example:

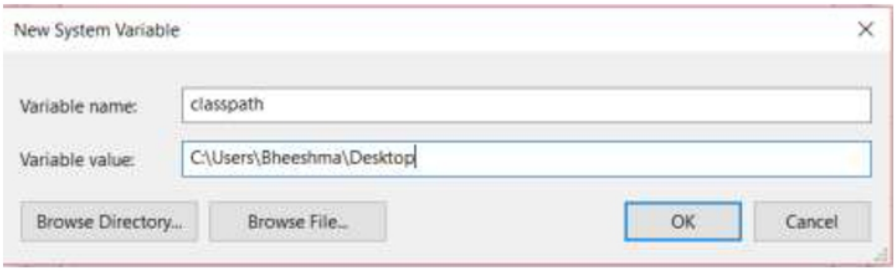
- Suppose JDK has been successfully installed in our PC. We have written a java program and saved in desktop. Now when we try to compile it using **javac** compiler, we may face error like below

```
C:\Users\Bheeshma\Desktop>javac Demo.java
'javac' is not recognized as an internal or external command,
operable program or batch file.
```

So to prevent from such error, operating system must be informed about the location of **javac** compiler and other libraries. This can be done using variable path as below



- After setting path system variable, we can successfully compile the java program.
- So when the file is compiled the object code i.e. .class file is created in the same location (in this case desktop). To execute it using **java** tool, we have to first navigate the command prompt to desktop. However, the task of navigation can be removed by setting **classpath** system variable as below.



- Now, we can run the object code also from the prompt other than desktop as shown below.

```
C:\Users\Bheeshma\Desktop>javac Demo.java

C:\Users\Bheeshma\Desktop>java Demo
Hellow World

C:\Users\Bheeshma\Desktop>d:

D:\>java Demo
Hellow World
```