Core Java Material

Chapter – 1

- Java Introduction
- Java Features
- Java Installation
- Java Program Execution Flow
- Java Programming Elements
- Java Programs Development
- Compilation
- Execution
- Translators
- JVM Architecture

What is Java

-> Java is a programming language

-> Java developed by James Gosling & his team in 1991 at Sun Microsystem Company

-> Initially they named it as ‘OAK’ programming language

-> In 1995, OAK language renamed to JAVA

Note: Oracle Corporation acquired Sun Microsystem in 2010. Now java is under license of Oracle corporation -> Java is a free & open-source software -> 3 billion devices run on

Java language

-> Java is one of the most popular languages in the world

-> Java can run on any platform (It is platform independent)

Java is divided into 3 parts

1) J2SE / JSE (Java Standard Edition) ➐ For stand-alone applications development

2) J2EE / JEE (Java Enterprise Edition) ➐ For Web applications development

3) J2ME / JME (Java Micro / Mobile Edition) ➐ For Mobile applications development

What we can develop using Java

-> Using Java, we can develop several kinds of applications like

1) Stand-alone applications
2) Web applications
3) Mobile Applications
4) Games
5) Servers
6) Databases and much more

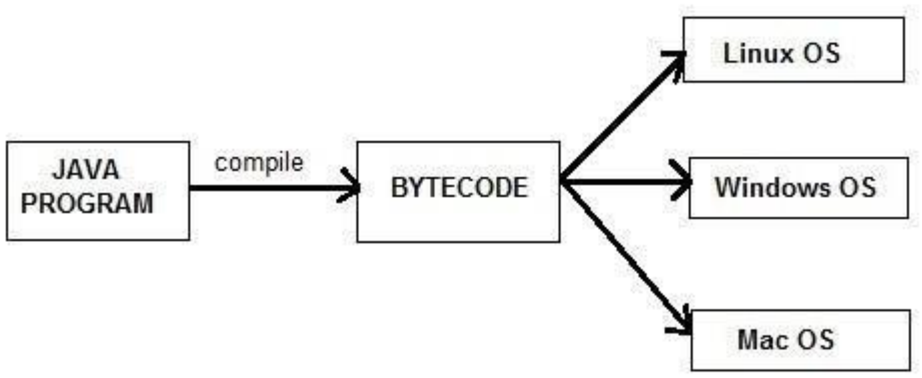
Java Features

1) **Simple:** Java is easy to learn and its syntax is quite simple, clean and easy to understand. The confusing and ambiguous concepts of C++ are either left out in Java or they have been re-implemented in a cleaner way.

Eg: Pointers and Operator Overloading are not there in java

2) **Platform Independent:** Unlike other programming languages such as C, C++ etc which are compiled into platform specific machines. Java is guaranteed to be writeonce, runanywhere language.

When we compile java code it will generate bytecode. This bytecode is platform independent and can be run on any machine, plus this bytecode format also provide security. Any machine with Java Runtime Environment can run Java Programs.



3) **OOP Language:** Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behaviour.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

OOPs Principles are:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

3) **Secure:** When it comes to security, Java is always the first choice. With java secure features it enables us to develop virus free, temper free system. Java program always runs in Java runtime environment with almost null interaction with system OS, hence it is more secure.

4) **Multi-Threading:** Java multithreading feature makes it possible to write program that can do many tasks simultaneously. Benefit of multithreading is that it utilizes same memory and other resources to execute multiple threads at the same time, like While typing, grammatical errors are checked along.

5) **Architectural Neutral:** Compiler generates bytecodes, which have nothing to do with a particular computer architecture, hence a Java program is easy to interpret on any machine.

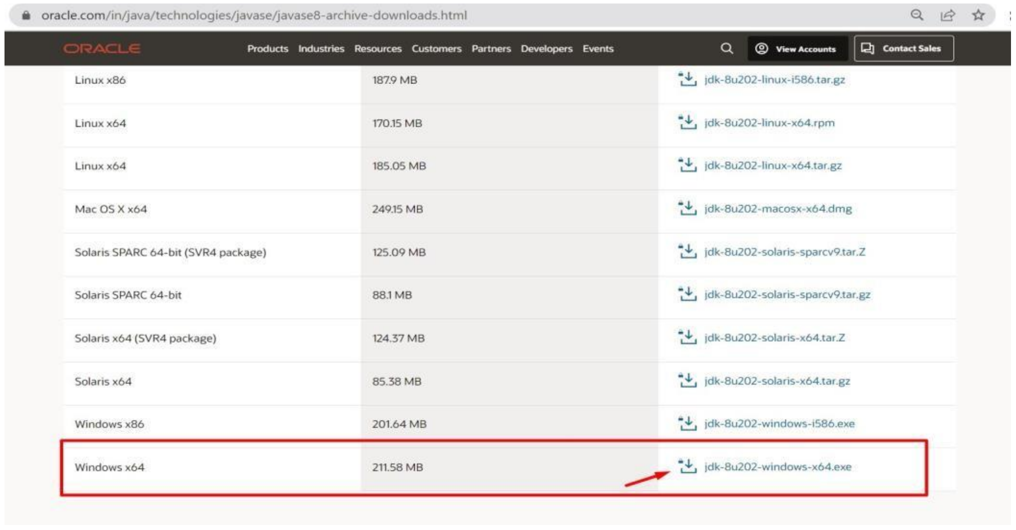
6) **Portable:** Java Byte code can be carried to any platform. No implementation dependent features. Everything related to storage is predefined, example: size of primitive data types

7) **High Performance:** Java is an interpreted language, so it will never be as fast as a compiled language like C or C++. But Java enables high performance with the use of just-intime compiler.

8) **Distributed:** Java is also a distributed language. Programs can be designed to run on computer networks. Java has a special class library for communicating using TCP/IP protocols. Creating network connections is very much easy in Java as compared to C/C++. **Java Slogan: WORA (Write Once Run Anywhere)**

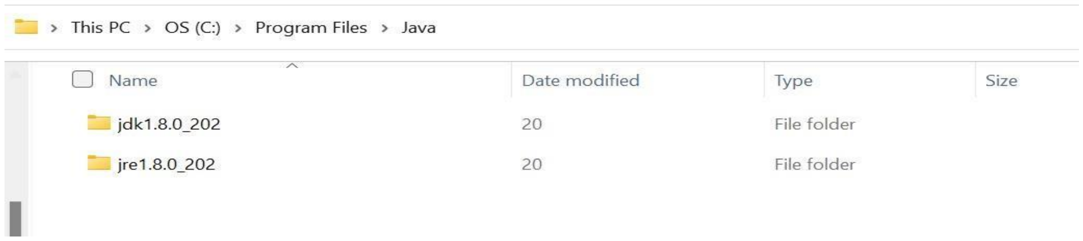
<u>Environment Setup (Java Installation)</u>

Step- 1) Download and Install Java Software



Linux x86	187.9 MB	jdk-8u202-linux-i586.tar.gz
Linux x64	170.15 MB	jdk-8u202-linux-x64.rpm
Linux x64	185.05 MB	jdk-8u202-linux-x64.tar.gz
Mac OS X x64	249.15 MB	jdk-8u202-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	125.09 MB	jdk-8u202-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	88.1 MB	jdk-8u202-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	124.37 MB	jdk-8u202-solaris-x64.tar.Z
Solaris x64	85.38 MB	jdk-8u202-solaris-x64.tar.gz
Windows x86	201.64 MB	jdk-8u202-windows-i586.exe
Windows x64	211.58 MB	jdk-8u202-windows-x64.exe

After installing java, we can see below two folders (JDK & JRE)



Name	Date modified	Type	Size
jdk1.8.0_202	20	File folder	
jre1.8.0_202	20	File folder	

Note: After installing java software, we need to set PATH for java software to use it.

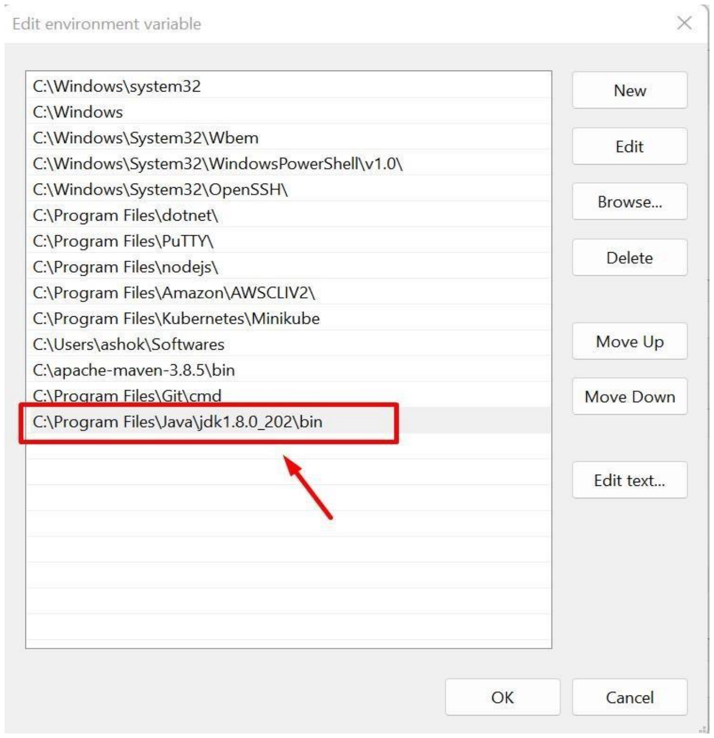
Step-2) Set Path for Java

-> Go To Environment Variables

-> Go To System Environment Variables

-> Edit Path

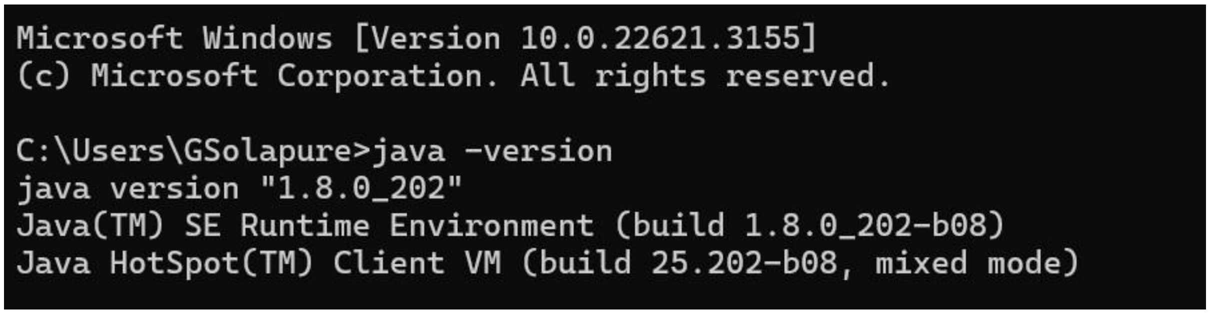
-> Add path up to JDK bin directory



```
C:\Windows\system32
C:\Windows
C:\Windows\System32\Wbem
C:\Windows\System32\WindowsPowerShell\v1.0\
C:\Windows\System32\OpenSSH\
C:\Program Files\dotnet\
C:\Program Files\PuTTY\
C:\Program Files\nodejs\
C:\Program Files\Amazon\AWSCLIV2\
C:\Program Files\Kubernetes\Minikube
C:\Users\ashok\Softwares
C:\apache-maven-3.8.5\bin
C:\Program Files\Git\cmd
C:\Program Files\Java\jdk1.8.0_202\bin
```

Path = C:\Program Files\Java\jdk1.8.0_202\bin

Step-3) Verify PATH Setup (Open command prompt and execute below command)



```
Microsoft Windows [Version 10.0.22621.3155]
(c) Microsoft Corporation. All rights reserved.

C:\Users\GSolapure>java -version
java version "1.8.0_202"
Java(TM) SE Runtime Environment (build 1.8.0_202-b08)
Java HotSpot(TM) Client VM (build 25.202-b08, mixed mode)
```

Note: If you are getting message like above that means, java installation and PATH setup is success.

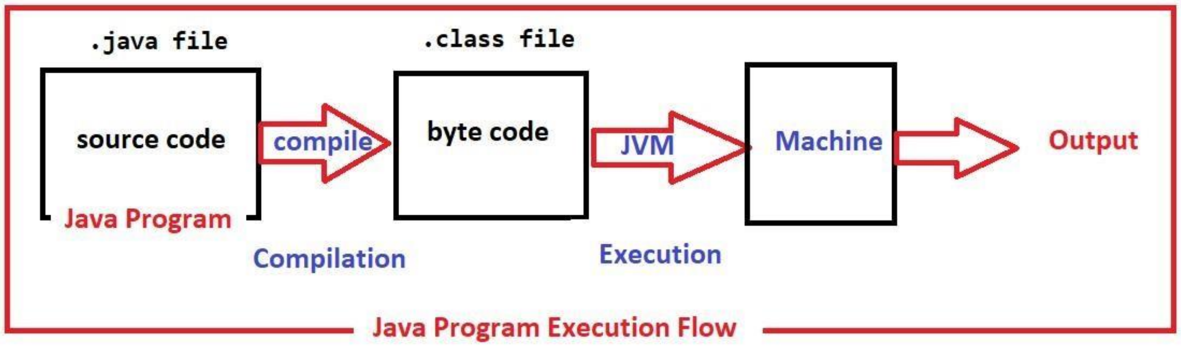
Q) What is the difference between JDK, JRE & JVM?

- JDK contains set of tools to develop java programs

- JRE providing a platform to run our java programs

- JVM will take care of program execution, memory allocation & de-allocation

Java Program Execution Flow

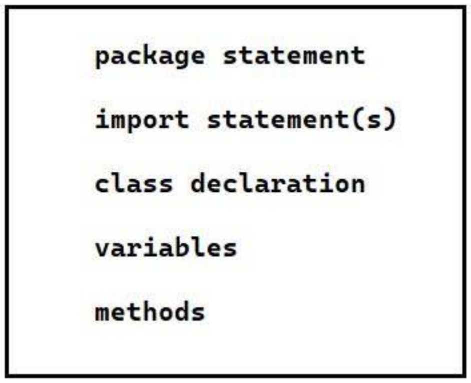


Step-1: We will write source code and we will save that code in a file using .java extension

Step-2: We will compile source code using java compiler (it will generate byte code)

Step-3: We will execute .class file (JVM will convert byte code into machine code & gives output)

Java Program Structure



```
package statement

import statement(s)

class declaration

variables

methods
```

-> **Package statement** is used to group our classes. At most we can write only one package statement in one java program and package statement should be the first statement in the program.

-> **Import statements** are used to import one program into another program. We can write multiple import statements in one program (It depends on requirement). We should write import statements after package statement only.

-> **Class** is the most import part in java program. Class is a plan to define program elements. Inside class we will write variables and methods.

-> **Variables** are used to store the data. We can write any no. of variables inside a class.

-> **Methods** are used to perform action. Application business logic we will write inside methods. A can class can have any no. of methods.

Developing First Java Program

Step-1: Open any text editor (notepad / notepad ++ / edit plus)

Step-2: Write below java program

```
public class hello { public static void

main(String[] args) {

  System.out.println("Hello World");

  }

}
```

Step-3: Save the program with .java extension

Step-4: Open Command prompt and change directory location to the place where our java program is available.

Step-5: Compile the Java program

Syntax:

- **javac filename.java**

We are able to see .class file that means our program compilation is successful.

Step-6: Run the java program

- **Java filename**

<u>Translators</u>

-> Translators are used to convert from one format to another format

-> We have 3 types of translators

1) Interpreter

2) Compiler

3) Assembler

-> Interpreter will convert the program line by line (performance is slow)

-> Compiler will convert all the lines of program at a time (performance is fast)

-> Assembler is used to convert assembler programming languages into machine language

<u>JVM</u>

-> JVM stands for Java Virtual Machine (We can't see with our eyes)

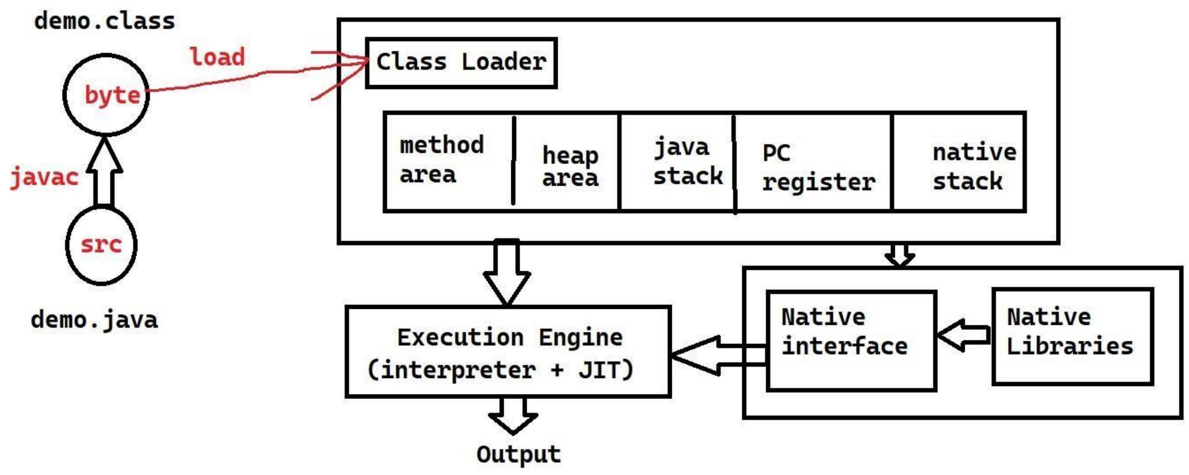
-> JVM will be part of JRE software

-> JVM is responsible for executing java programs

-> JVM will allocate memory required for program execution & de-allocate memory when it is not used

-> JVM will convert byte code into machine understandable format

<u>JVM Architecture</u>



Class loader: It will load .class file into JVM

Method Area: Class code will be stored here

Heap area: Objects will be stored into heap area

Java Stack: Method execution information will be stored here

PC Register: It will maintain next line information to execute

Native Stack: It will maintain non-java code execution information

Native Interface: It will load native libraries into JVM

Native Libraries: Non-java libraries which are required for native code execution

Execution Engine: It is responsible to execute the program and provide output/result. It will use Interpreter and JIT for execution.