ShreeSoft INFORMATICS

Address: Tirumala Plaza, Upendra Nagar, Cidco, Nashik.

Join us on Telegram:
https://t.me/ShreeSoftinformatics

Join us on WhatsApp:
https://chat.whatsapp.com/BrLM0ZoWMbpA8nXi9Mrocl

Contact: +91 8308341531

Introduction to Java Programming Language

Definition:

Java is Programming suite because java is both a **programming language** and **a platform**. Java is a general-purpose, class-based, object-oriented programming language.

What is Platform?

Java is a platform-independent language because it has runtime environment i.e. JRE and API. Here platform means a hardware or software environment in which application runs.

History of java

Initially Java was designed for Interactive television, but this technology was very much advanced for the industry of digital cable television at that time. Java history was started with the Green Team. The Green Team started a project to develop a language for digital devices such as television. But it works best for internet programming. After some time Java technology was joined by Netscape. Initially, Java was called "Greentalk" by James Gosling and at that time the file extension was .gt

Author	: James Gosling
Vendor	: Sun Micro System
Project name	: Green Project
Type	: open source & free software
Initial Name	: OAK language
Present Name	: java
Extensions	: .java & .class & .jar
Initial version	: jdk 1.0 (java development kit)
Present version	: java 19
Operating System	: multi Operating System
Implementation Lang	: C & C++
Symbol	: coffee cup with saucer
Objective	: To develop web applications
SUN	: Stanford Universally Network
Slogan/Motto	: WORA (write once run anywhere)
Currently Acquired by	: Oracle Corporation

Application areas of Java

1. **Dekstop Application**
2. **Web Application**
3. **Enterprise Application**
4. **Mobile Application**
5. **Embedded System**
6. **Game Development**
7. **Distributed Application**
8. **Robotics**
9. **Smart Card**

Features of Java

1. **Simple:** Easy to learn, most of the difficult features like pointers, multiple inheritance concepts have been removed from Java.
2. **Platform independent:** Write once run anywhere.
3. **Architecture Neutral:** If you perform any changes in your architecture of system, java program will still run without any issue.
4. **Portable:** We can migrate java code from windows to Linux easily.
5. **Secure:** Lets you creating applications that can't be invaded from outside.
6. **Object Oriented:** From JDK 1.8 onwards Java has also incorporated functional (procedural) programming.
7. **Multi-threaded:** You can build applications with many concurrent threads of activity, resulting in highly responsive applications.
8. **Robust:** Having reliable programming habits for creating highly reliable applications.
9. **Distributed:** It facilitates users to create distributed applications in Java. To create distributed applications we use RMI, EJB, CORBA etc.
10. **Interpreted:** Java integrates the power of Compiled Languages with the flexibility of Interpreted Languages.
11. **High Performance:** Java code after compilation converts to bytecode which is highly optimized by the Java compiler, so that the Java virtual machine (JVM) can execute Java applications at full speed.
12. **Dynamic:** It supports dynamic loading of classes. It means class loading happens on demand.

How is Java a platform independent language?

In general the compiler's job in a programming language is to convert source code into machine understandable code (also called executable code). There is one more state of code between source code & machine understandable code in java which is called the byte-code. In fact converting java source code to machine understandable code is two-step process. One is from source code to byte-code (.class file) which is done by compiler. Other is from byte-code to machine understandable code which is done by the JVM. However JVM is a software which comes automatically with JDK installation.

JVM is platform dependent. Therefore while converting from byte-code to executable code JVM makes it compatible with the platform to which it belongs. In this complete process there are two translators which makes this possible ie. Compiler & JVM. Compiler can read source file (.java file). Similarly JVM can read byte-code (.class file). Because of the feature of platform in-dependency Java's slogan in **Write Once Run Anywhere (WORA)**. Also below points are important to keep in mind.

Java Source Code - platform independent
Java Bytecode - platform independent
Java Compiler - platform independent
Java Executable Code - platform dependent
Java Virtual Machine - platform dependent
Java Software (jdk) - platform dependent
Java Program - platform independent
Java Software Application - platform independent

The languages where conversion of source code to machine language is a single process, those languages come under platform dependent category like C, C++.

Different Types of Java Platforms
There are four different types of Java programing language platforms:

Java Standard Edition (Java SE / J2SE):
Java SE's API offers the Java programming language's core functionality. It defines all the basis of type and object to high-level classes. It is used for networking, security, database access, graphical user interface (GUI) development, and XML parsing.

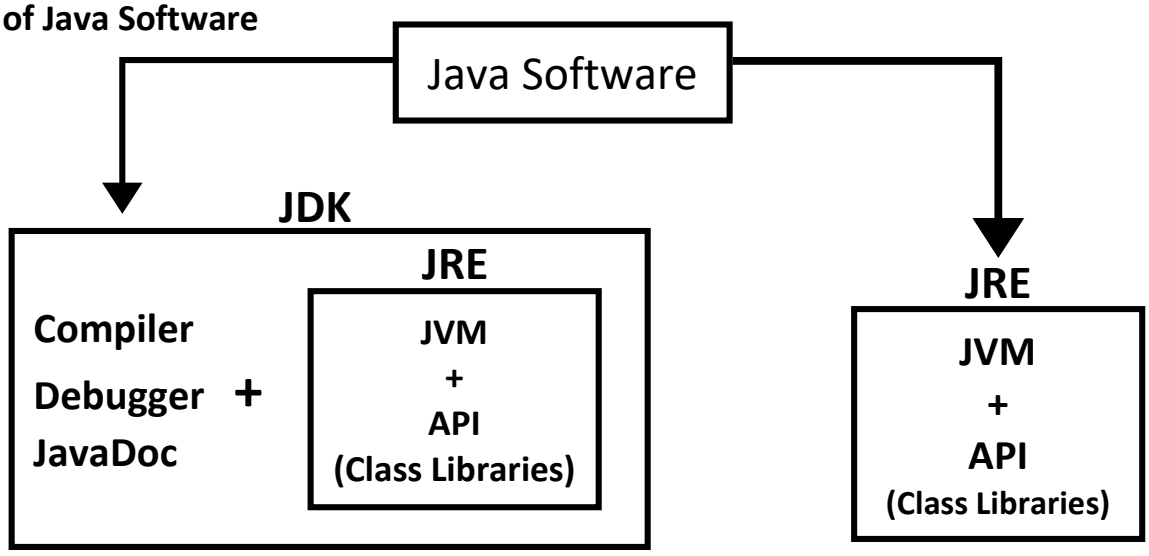
Java Enterprise Edition (Java EE / J2EE):
The Java EE platform offers an API and runtime environment for developing and running highly scalable, large-scale, multi-tiered, reliable, and secure network applications.

Java Micro/Mobile Edition (Java ME / J2ME):
The Java ME platform offers an API and a small-footprint virtual machine running Java programming language applications on small devices like mobile phones.

Java FX:
JavaFX is a platform for developing rich internet applications using a lightweight user-interface API. It user hardware-accelerated graphics and media engines that help Java take advantage of higher-performance clients and a modern look-and-feel and high-level APIs for connecting to networked data sources.

Two Types of Java Software

1) **JDK**
2) **JRE**



What is JDK?

JDK is a software development environment used for making applets and Java applications. The full form of JDK is Java Development Kit. JDK is a platform-specific software and that's why we have separate installers for Windows, Mac, and UNIX systems. JDK helps them to code and run Java programs. It is possible to install more than one JDK version on the same computer.

Why use JDK?

- JDK contains tools required to write Java programs, and JRE to execute them.
- It includes a compiler, Java application launcher, Appletviewer etc.
- Compiler converts code written in Java into byte code.
- Java application launcher opens a JRE, loads the necessary class, and executes its main method.

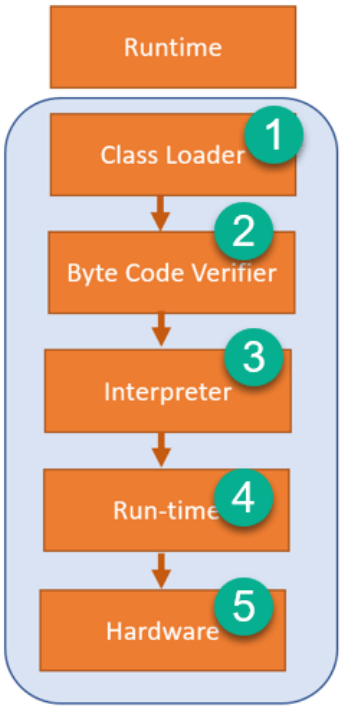
What is JRE?

JRE is a piece of a software which is designed to run other software. It contains the class libraries, loader class, and JVM. In simple terms, if you want to run Java program you need JRE. If you are not a programmer, you don't need to install JDK, but just JRE to run Java programs. Though, all JDK versions comes bundled with Java Runtime Environment, so you do not need to download and install the JRE separately in your PC. The full form of JRE is Java Runtime Environment.

Why use JRE?

- JRE contains class libraries, JVM, and other supporting files. It does not contain any tool for Java development like a debugger, compiler, etc.
- It uses important package classes like math, swing, util, lang, awt, and runtime libraries.
- If you have to run Java applets, then JRE must be installed in your system.

How JRE Works?



JRE

JRE has an instance of JVM with it, library classes, and development tools. Once you write and compile Java code, the compiler generates a class file having byte code. Here are the important components of JRE:

- **Class loaders:** The class loader loads various classes that are necessary for running a Java program. JVM uses three class loaders called the bootstrap class loader, extensions class loader, and system class loader.
- **Byte code verifier:** Byte code verifier verifies the bytecode so that the code doesn't disturb the interpreter.
- **Interpreter:** Once the classes get loaded, and the code is verified, the interpreter reads the code line by line.
- **Run-time:** Run-time is a system used mainly in programming to describe time period during which a particular program is running.
- **Hardware:** Once you compile Java native code, it runs on a specific hardware platform.

What is JVM?

JVM is a very important component of Java programming language. When you run the Java program, the Java compiler first compiles your Java code to bytecode. Then, the JVM translates bytecode into native machine code (set of instructions that a computer's CPU executes directly).

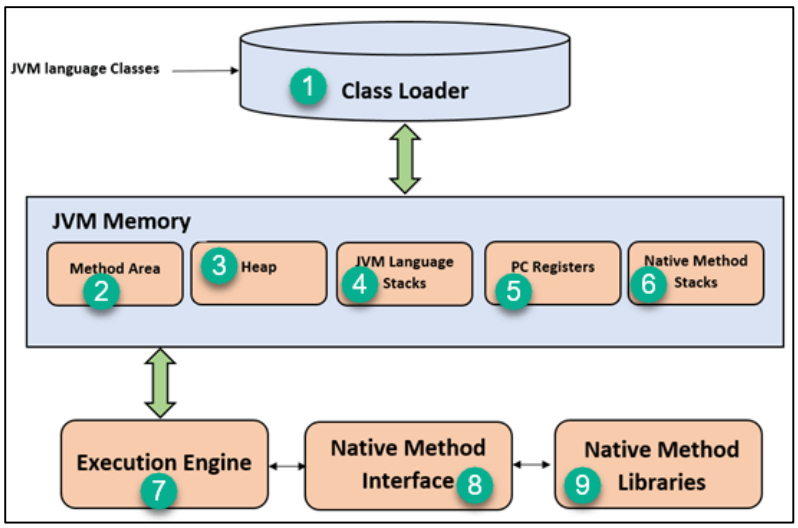
JVM translates the byte-code into machine language, and since the machine language depends on the operating system being used, it is clear that the JVM is platform (operating system) dependent.

JVM is called virtual because it provides an interface that does not depend on the underlying operating system and machine hardware.

Why JVM?

Here are the important reasons of using JVM:
JVM provides a platform-independent way of executing Java source code.
It has numerous libraries, tools, and frameworks.
Once you run Java program, you can run on any platform and save lots of time.
JVM comes with JIT (Just-in-Time) compiler that converts Java source code into low-level machine language. Hence, it runs faster as a regular application.

How JVM Works?



1) Class Loader: The class loader is a subsystem used for loading class files. It performs three major functions viz. Loading, Linking, and Initialization.

2) Method Area: JVM Method Area stores structure of class like metadata, the code for Java methods, and the constant runtime pool.

3) Heap: All the Objects, arrays, and instance variables are stored in a heap. This memory is shared across multiple threads.

4) JVM language Stacks: Java language Stacks store local variables, and its partial results. Each and every thread has its own JVM language stack, created concurrently as the thread is created. A new frame is created when method is invoked, and it is removed when method invocation process is complete.

5) PC Registers
PC registers store the address of the Java virtual machine instruction, which is currently executing. In Java, each thread has its separate PC register.

6) Native Method Stacks
Native method stacks hold the instruction of native code depends on the native library. It allocates memory on native heaps or uses any type of stack.

7) Execution Engine
It is a type of software that is used to test software, hardware, or complete systems. The test execution engine never carries any information about the tested product.

8) Native Method interface
The Native Method Interface is a programming framework. It allows Java code, which is running in a JVM to call by libraries and native applications.

9) Native Method Libraries
Native Libraries is a collection of the Native Libraries (C, C++), which are needed by the Execution Engine.