Vinit Raj

Oct 16 · 5 min read · Listen

Save

All about JVM (Java Virtual Machine)

Java Virtual Machine

When it comes to gaining ‘true’ knowledge about some technology, nothing beats it's official documentation. So, here is the link. I have tried to write a summary of information about JVM available on various websites.

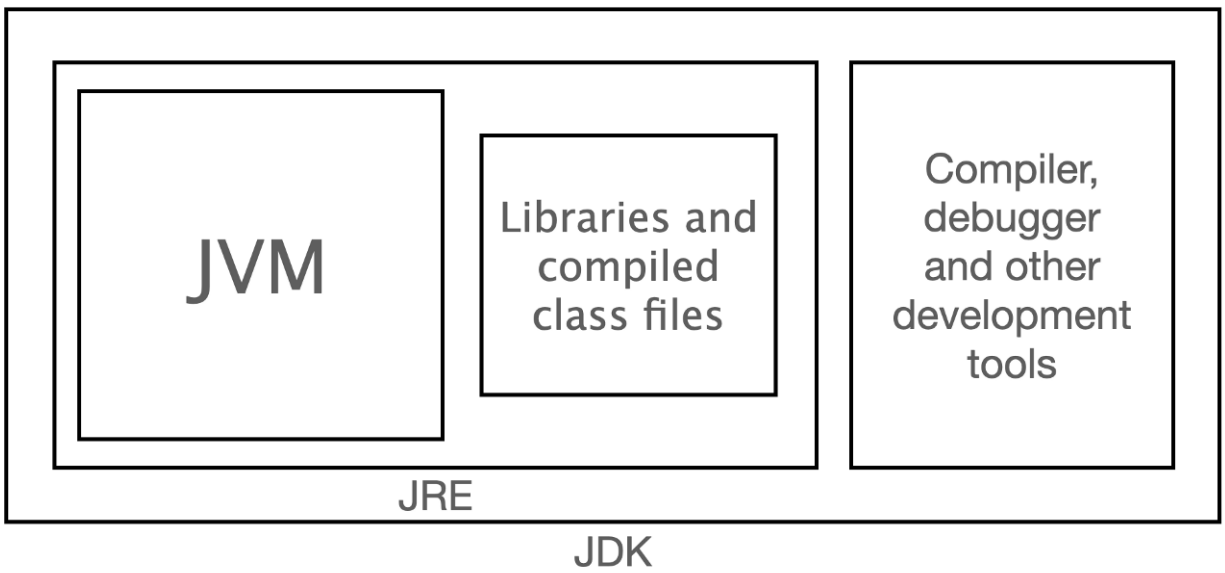
What is JVM?

JVM or **Java Virtual Machine** is a virtual machine that enables an user to run code on a system written in *Java, Scala, Kotlin, Clojure, Jython, JRuby, Groovy* or any other

Where did JVM come from?

The first prototype implementation of the Java Virtual Machine, done at Sun Microsystems Inc., emulated the Java Virtual Machine instruction set in software hosted by a handheld device that resembled a contemporary Personal Digital Assistant (PDA).

Basically, Java has three components, viz.
i. Java Development Kit (JDK)
ii. Java Runtime Environment (JRE)
iii. Java Virtual Machine (JVM)



JDK, JRE, JVM

Out of these three, our primary focus, for now, is on Java Virtual Machine.

Every code we write, needs a system to run on. At the time Java was developed, there was an issue. The issue was, machine code of executable file needed to be in a format that was understood by the processor of the system. Different platforms or different brands of processor needed different machine code of executable file.

> *The idea was to develop a standard machine which could run executable file of Java Code on any platform irrespective of kind of processor without having to re-compile it.*

Keeping this in mind, a virtual machine was developed which could run on any system irrespective of brand of processor or operating system, and this 'virtual' machine could execute command(s) of code. Now the hustle of compiling different codes for different platforms or O.S. was gone. A machine being able to run Java code on any platform without having to re-compile the code file! This virtual system is Java Virtual Machine (JVM).

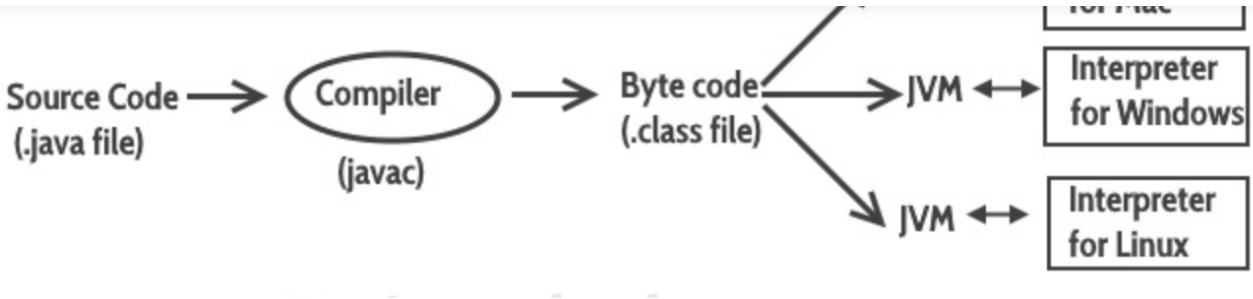
> *YOU WRITE THE CODE ONCE.*
> *YOU COMPILE THE CODE ONCE.*
> *YOU EXECUTE THE CODE ONCE.*
> *YOU RUN IT ON ANY MACHINE ANYTIME ANYWHERE !*

Each operating system has different JVM, however the output they produce after execution of byte code is same across all operating systems. Hence JVM makes Java portable, Write Once Run Anywhere (WORA).
JVM is the one that makes Java platform independent.

> ***There are basically two types of implementation of JVM — Client** and **Server. Every company has its own JVM implementation like Oracle has HotSpot JVM, IBM has J9 JVM and so on.***

What exactly does JVM run?
Machine language for JVM is byte code. JVM runs byte code (complied machine code of Java) or .class file produced after compilation of Java code.
*In the block of code, **class** is / are present and class is a blueprint for an object. Class tells the virtual machine how to make an object of that particular type.*



Flow chart of how Java code is compiled and run.

Use of JVM

JVM converts bytecode (.class file in case of Java) to machine specific code. It is platform dependent and performs many functions like memory management and security. Also, JVM enables a computer to run Java programs as well as programs written in other languages that are also compiled to Java bytecode.

Modes or implementation of JVM

Depending on the organisation, there can be different versions and implementations of JVM, like for example Oracle has two different versions of JVM, viz. ***Java Hotspot Client VM*** and ***Java Hotspot Server VM.***

The client VM is tuned for reducing start-up time and memory footprint.

The server VM is designed for maximum program execution speed.

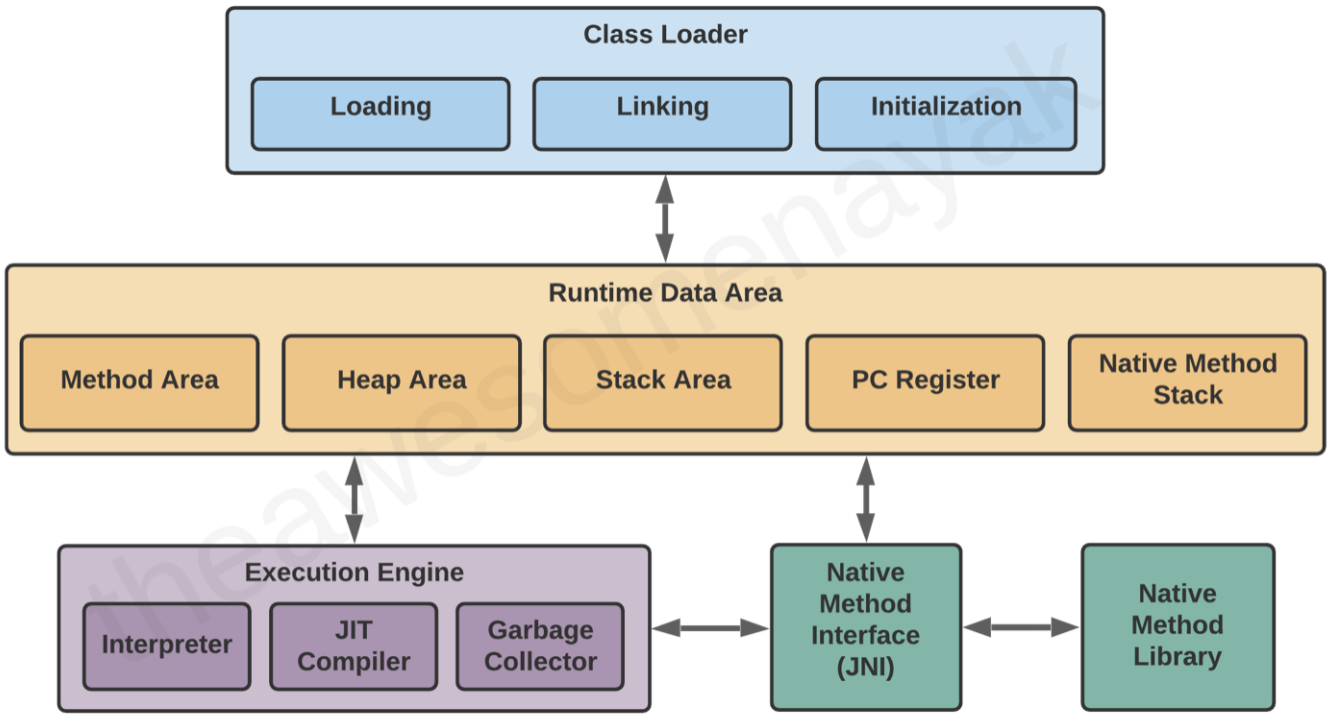
JVM lives to run one Java application and dies after the application completes.

On one computer if three Java applications are started then three JVMs are started seperately. The do not share anything in common on OS.

Java Virtual Machine Architecture

(Here is the link of official documentation.)

ii. Runtime Memory/Data Area
iii. Execution Engine



Java Virtual Machine Architecture design

Class Loader

When you compile a *.java* source file, it is converted into byte code as a *.class* file. The class loader reads the .class file and save the byte code in the **method area** (of Runtime Data Area).

Method Area

There is only one method area in a JVM which is shared among all the classes. This holds the class level information of each .class file.
If the memory available in the method area is not sufficient for the program startup, the JVM throws an *OutOfMemoryError* .

Heap Area

Heap is a part of JVM memory where objects are allocated. JVM creates a Class object for each .class file. All the objects and their corresponding instance variables are stored here. This is the run-time data area from which memory for all class instances and arrays is allocated.

temporary variables.

Program Counter (PC) Register
The JVM supports multiple threads at the same time. Each thread has its own PC Register to hold the address of the currently executing JVM instruction. This keeps a track of the instructions that have already been executed as well as the instructions that are going to be executed. Since instructions are executed by threads, each thread has a separate PC register.

Native Method Stack
The JVM contains stacks that support *native* methods. These methods are written in a language other than Java, such as C and C++. For every new thread, a separate native method stack is also allocated.

Execution Engine
Once the bytecode has been loaded into the main memory, and details are available in the runtime data area, the next step is to run the program. The Execution Engine handles this by executing the code present in each class.

Java Native Method Interface (JNI)
It is necessary to use native (non-Java) code, for example C or C++. This can be in cases where we need to interact with hardware, or to overcome the memory management and performance constraints in Java. Java supports the execution of native code via the Java Native Interface (JNI).

Native Method Library
Native Method Libraries are libraries that are written in other programming languages, such as C, C++, and assembly. These native libraries can be loaded through JNI.

Thank you!

References :




architecture-explained-for-beginners/
https://www.tutorialspoint.com/java_virtual_machine/java_virtual_machine_quick_guide.htm
https://www.geeksforgeeks.org/jvm-works-jvm-architecture/