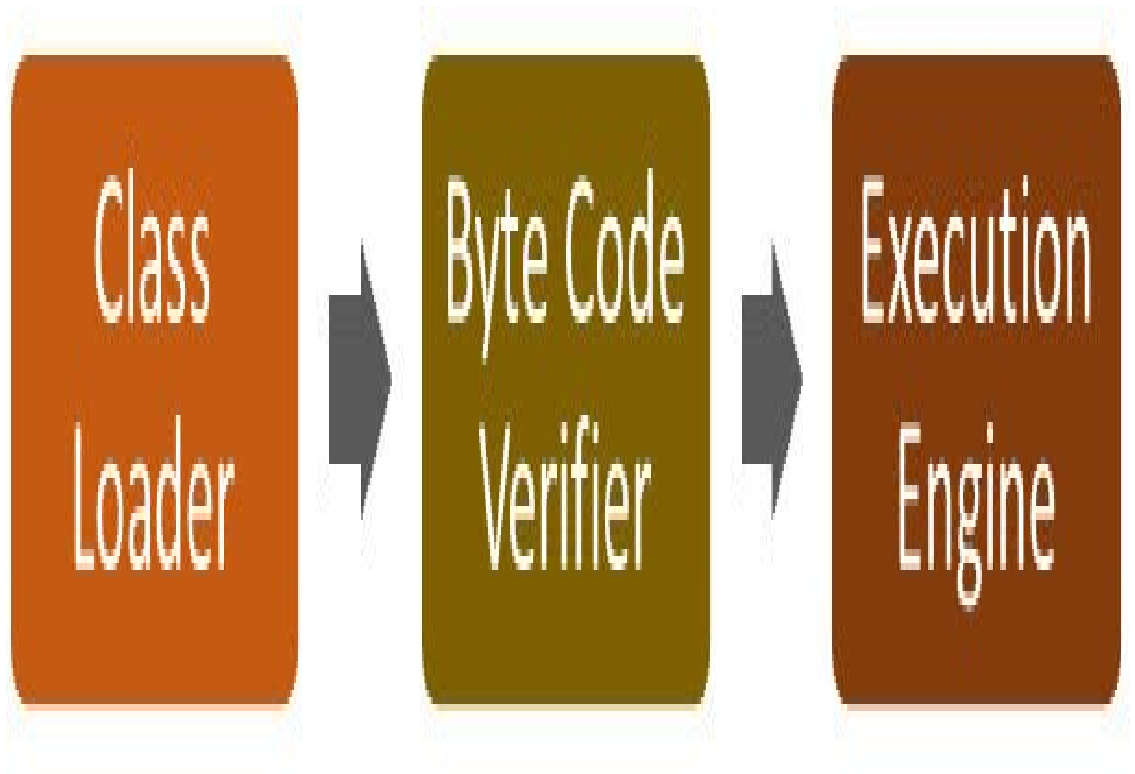
Java Virtual Machine (JVM) & its Architecture

What is JVM?

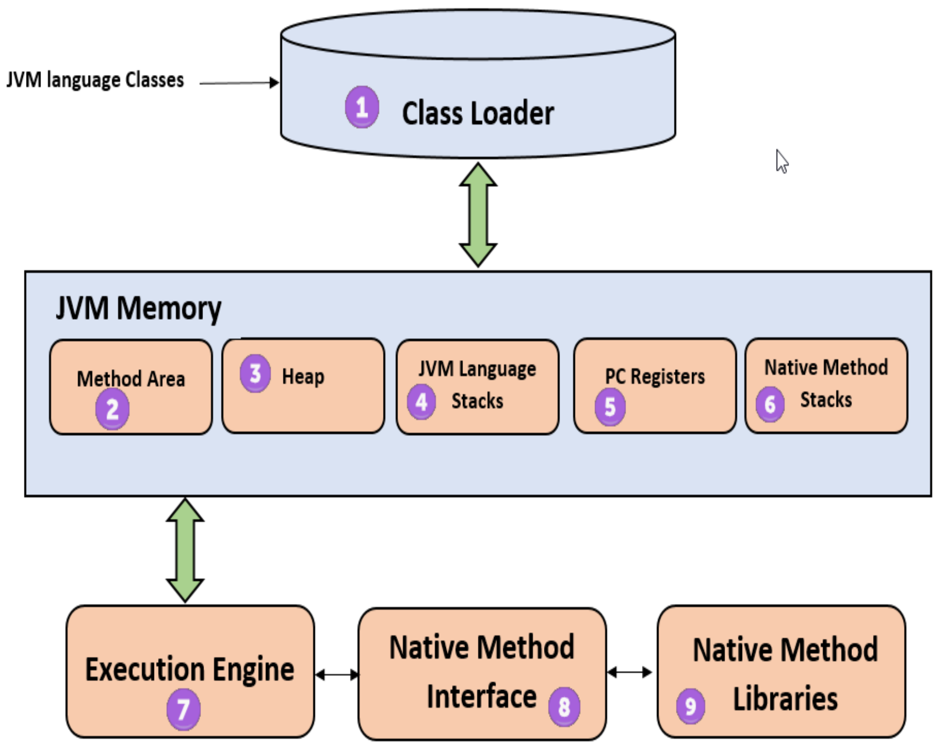
JVM is a engine that provides runtime environment to drive the Java Code or applications. It converts Java bytecode into machines language. JVM is a part of JRE(Java Run Environment). It stands for Java Virtual Machine

- In other programming languages, the compiler produces machine code for a particular system. However, Java compiler produces code for a Virtual Machine known as Java Virtual Machine.
- First, Java code is complied into bytecode. This bytecode gets interpreted on different machines
- Between host system and Java source, Bytecode is an intermediary language.
- JVM is responsible for allocating memory space.



JVM Architecture

Let's understand the Architecture of JVM. It contains classloader, memory area, execution engine etc.



- **ClassLoader**

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

- **Method Area**

JVM Method Area stores class structures like metadata, the constant runtime pool, and the code for methods.

- **Heap**

All the Objects, their related instance variables, and arrays are stored in the heap. This memory is common and shared across multiple threads.

- **JVM language Stacks**

Java language Stacks store local variables, and it's partial results. Each thread has its own JVM stack, created simultaneously as the thread is created. A new frame is created whenever a method is invoked, and it is deleted when method invocation process is complete.

- **PC Registers**

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

- **Native Method Stacks**

Native method stacks hold the instruction of native code depends on the native library. It is written in another language instead of Java.

- **Execution Engine**

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

- **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.

- **Native Method Libraries**

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

Software Code Compilation & Execution process

In order to write and execute a software program, you need the following

- **Editor** – To type your program into, a notepad could be used for this
- **Compiler** – To convert your high language program into native machine code
- **Linker** – To combine different program files reference in your main program together.
- **Loader** – To load the files from your secondary storage device like Hard Disk, Flash Drive, CD into RAM for execution. The loading is automatically done when you execute your code.
-

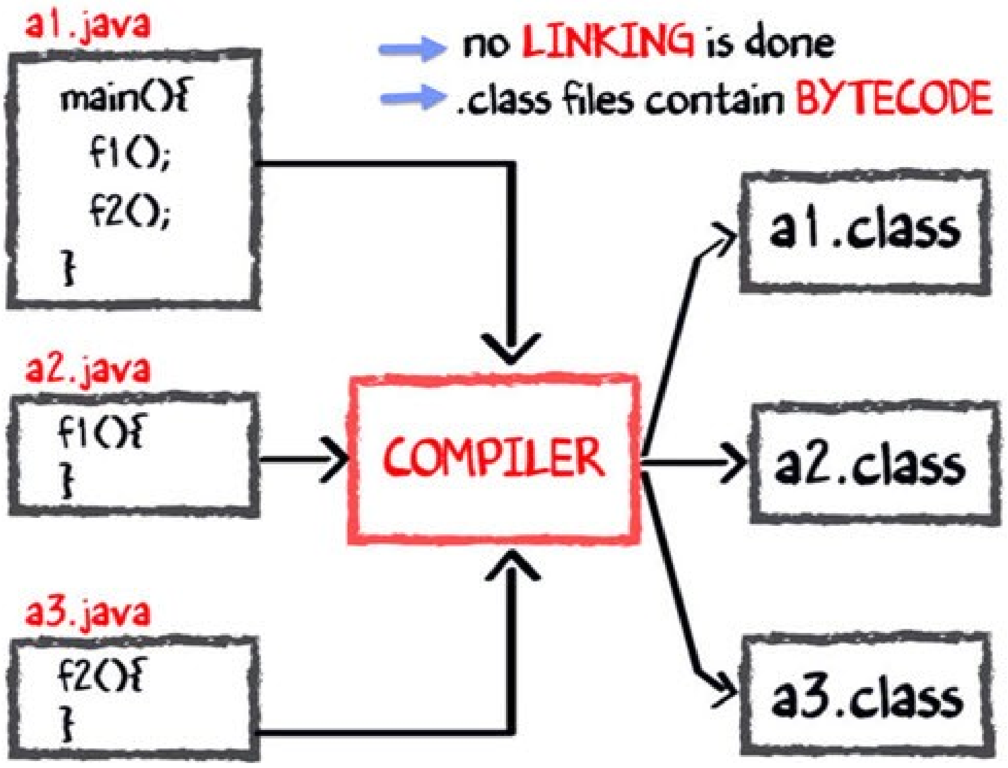
Execution – Actual execution of the code which is handled by your OS & processor.

With this background, refer the following video & learn the working and architecture of the Java Virtual Machine.

Java code Compilation and Execution in Java VM

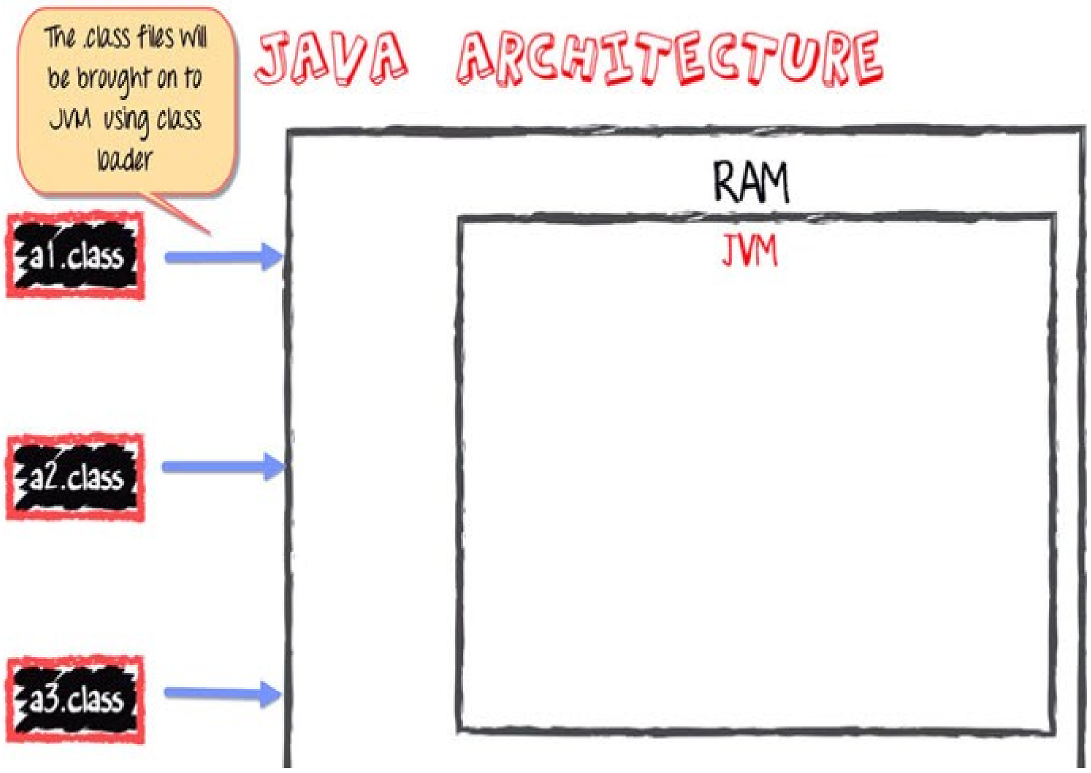
Let's look at the process for JAVA. In your main, you have two methods f1 and f2.

- The main method is stored in file a1.java
- f1 is stored in a file as a2.java
- f2 is stored in a file as a3.java

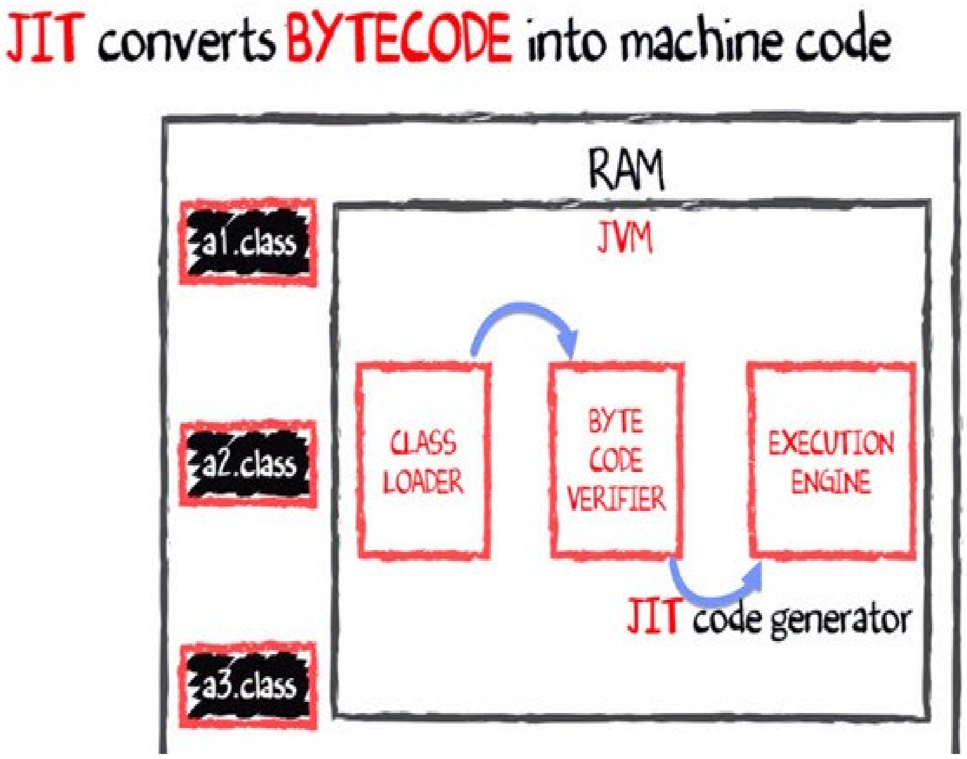


The compiler will compile the three files and produces 3 corresponding .class file which consists of BYTE code. Unlike C, no linking is done.

The Java VM or Java Virtual Machine resides on the RAM. During execution, using the class loader the class files are brought on the RAM. The BYTE code is verified for any security breaches.


The .class files will be brought on to JVM using class loader
JAVA ARCHITECTURE
RAM
JVM
a1.class
a2.class
a3.class

Next, the execution engine will convert the Bytecode into Native machine code. This is just in time compiling. It is one of the main reason why Java is comparatively slow.



NOTE: JIT or Just-in-time compiler is the part of the Java Virtual Machine (JVM). It interprets part of the Byte Code that has similar functionality at the same time.

Why is Java both Interpreted and Compiled Language?

Programming languages are classified as

- Higher Level Language Ex. C++, Java
- Middle-Level Languages Ex. C
- Low-Level Language Ex Assembly
- finally the lowest level as the Machine Language.

A **compiler** is a program which converts a program from one level of language to another. Example conversion of C++ program into machine code.

The java compiler converts high-level java code into bytecode (which is also a type of machine code).

An **interpreter** is a program which converts a program at one level to another programming language at the same level. Example conversion of Java program into C++

In Java, the Just In Time Code generator converts the bytecode into the native machine code which are at the same programming levels.

Hence, Java is both compiled as well as interpreted language.

Why is Java slow?

The two main reasons behind the slowness of Java are

- Dynamic Linking: Unlike C, linking is done at run-time, every time the program is run in Java.
- Run-time Interpreter: The conversion of byte code into native machine code is done at run-time in Java which furthers slows down the speed

However, the latest version of Java has addressed the performance bottlenecks to a great extent.

Summary:

- JVM or Java Virtual Machine is the engine that drives the Java Code. It converts Java bytecode into machines language.
- In JVM, Java code is compiled to bytecode. This bytecode gets interpreted on different machines

- JIT or Just-in-time compiler is the part of the Java Virtual Machine (JVM). It is used to speed up the execution time
- In comparison to other compiler machines, Java may be slow in execution.