**Java Virtual Machine(JVM)**

**What is Java virtual Machine ?**

* Java Virtual machine is a Runtime responsible to run java based applications .
* It has two main tasks :

1. Load .class file
2. Execute .class file

* Basic Architecture of JVM is :



* **Class Loader** : It is responsible for following three tasks –

1. Loading
2. Linking
3. Initialization

**Class Loader Subsystem**

**Loading :**

* It will read .class file and store corresponding information in the method area .
* For each class file , JVM will store following information in the method area :

1. Fully qualified class name
2. Fully qualified parent class name
3. Methods information
4. Variables information
5. Modifiers information
6. Constant pool information

* Three types of class loaders :

1. Bootstrap class loader : Responsible to load the classes present in rt.jar (rt.jar is present in bootstrap class path i.e. jdk / jre / lib)
2. Extension class loader : Responsible to load the classes from extension class path (jdk / jre / lib / ext /\*.jar)
3. Application class loader : Responsible to load the classes from application class path .It internally uses environment variable class path .

**Linking :**

In linking three activities are performed :

1. Verification
2. Preparation
3. Resolution
4. **Verification :** It is a process Byte code Verifier checks whether the .class file is generated by valid compiler or not and whether .class file properly formatted or not .

If verification fails , then JVM will provide “java.lang.VerifyError” exception . Because of this process , java is secured .

1. **Preparation** : In this process JVM will allocate memory for class level static variables & assign default values (not original values)
2. **Resolution :** In this process symbolic names present in our program are replaced with original memory references from method area .

**Initialization :**

In this process , two activities will be performed :

1. All static variables are assigned with original values .
2. Static blocks will be executed from top to bottom

**Memory Areas**

Total 5 types of memory areas –

1. Method Area
2. Heap Area
3. Stack Area
4. PC Register
5. Native Method Area
6. **Method Area –**

* Method area is created when JVM is started .
* It stores .class file information and static variable
* Per JVM one memory area , therefore multiple threads can access this area , so it is not thread safe .

1. **Heap Area -**

* Heap area is created when JVM is started .
* It store objects , instance variable and arrays (as every arrays is an object in java) .
* It can be accessed by multiple threads , so the data stored in heap area is not thread safe .

1. **Stack Area –**

* Whenever a new thread is created , a separate stack area will also be created
* It stores the current running method and local variables .
* When the method is completed , the corresponding entry from stack will be removed .
* After completing all method calls , the stack will become empty and that empty stack will be destroyed by the JVM just before terminating the thread
* The data stored in the stack is available only for the corresponding thread and not available to the remaining thread , so this area is thread safe .

1. **PC Register –**

* It holds the address of next executing instruction .
* For every thread , a separate pc register is created , so it is also thread safe .

1. **Native Method Stacks** –

* All Native method calls invoked by the thread will be stored in the corresponding native method stack .
* For every thread separate native method stack will be created
* It is also thread safe .

**Execution Engine**

Execution engine is responsible to execute java class file .

It contains mainly two components :

1. Interpreter
2. JIT Compiler

**Interpreter -**

A module that alternately decodes and executes every statements or line in some body of code . The java interpreter decodes and executes bytecode for the java virtual machine .

**Byte code → interpret → machine code or native code → execute**

**JIT compiler -**

* JIT stands for Just-In-Time which means that code gets compiled when it is needed , not before runtime .
* The main purpose of JIT compiler is to improve performance .
* JVM maintain a count as of how many time a function is executed . If this count exceeds a predefined limit or say threshold value , the JIT complies the code into machine language which can directly be executed by the processor (unlike normal case in which javac compile the code into bytecode and then java – the interpreter interprets this bytecode line by line converts into machine code and executes) .
* JIT compilation is applicable only for repeatedly required methods , not for every method .

**Java Native Interface(JNI)**

* An interface that allows java to interact with code written in another language.
* It acts as mediator for java methods calls & the corresponding native libraries i.e. JNI is responsible to provide information about native libraries to the JVM .
* Native Method Library provides or holds native libraries information.
* The java command – line utility is an example of one such application , that launches java code in a Java Virtual Machine .