**JVM (Java Virtual Machine)**

**Virtual Machine (VM)**

> Software stimulation of a machine which can perform operations like physical machine is called Virtual Machine.

> 2 Categories of VM with respect to Programming:

1. Hardware based VM/ System based VM

2. Application based VM/ Process based VM

1. Hardware based VM: - It provides several logical systems on the same computer with the strong isolation from each other i.e. on one physical machine; we are defining multiple logical machines.

> Main Advantage of H/w based VM is H/w resources utilization & to improve utilization of H/w resources.

For E.g. KVM (Kernel based VM) for Linux system, VMWARE, Cloud Computing etc.

2. Application based VM: - These VM machines act as runtime engines to run a particular programming language application.

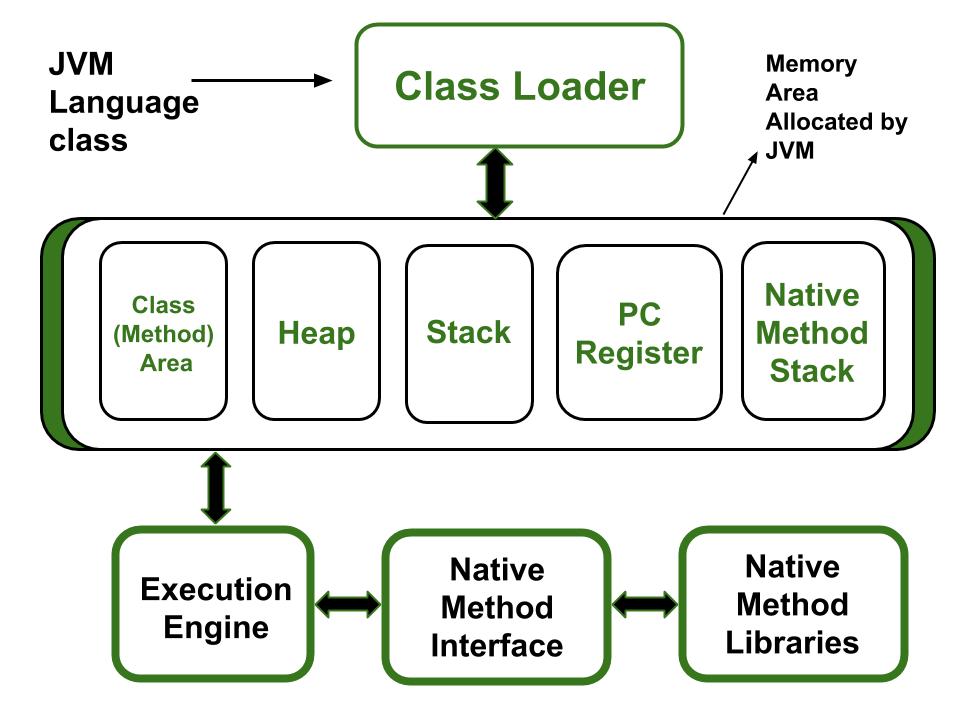
For E.g. JVM acts as runtime engine to run Java based applications.

PVM acts as runtime engine to run Perl based applications.

CLR acts as runtime engine to run .NET based applications.

> JVM is a part of JRE (part of JDK)

.class file Class Loader Subsystem



**Class loader subsystem** reads .class file & store it inside JVM Memory. **Execution Engine** is responsible for reading the .class file & executing it which results in displaying the corresponding output to the console.

**Class Loader Subsystem**

> It is responsible for 3 activities

1. Loading

2. Linking

3. Initialization

1. **Loading**: - Loading means reading class file & store the corresponding binary data in Method Area.

> For each class file, JVM will store corresponding information in the method area like Fully Qualified name of class, immediate parent class, Method information, variable information, constructors info, modifiers info, constant pool info etc.

> After loading .class file, JVM immediately creates an object for that loader class on the heap memory of type **java.lang.Class** (object called as Class class object).

> This Class class object can be used by Programmer to get class level information like method info, variable info or constructor info etc.

> Refer programs from repository.

2. **Linking**: - It mainly contain 3 activities

a) **Verify**: - It is the process of ensuring that binary representation of a class is structurally correct or not (formatted or not) i.e. JVM will check whether the .class file generated by a valid compiler or not.

> Internally, **Bytecode verifier** is responsible for this activity & it is a part of class loader subsystem.

> If verification fails, then we will get runtime exception saying **java.lang.verifyError.**

> **This is why Java is Secure.**

b) **Prepare**: - In this process, JVM will allocate memory to the class level static variables & assign default values (Not original value).

c) **Resolve**: - It is the process of replacing symbolic names in our program with original memory references from method area.

3. **Initialization**: - In this activity, all static variables are assigned with original values & static blocks will be executed from parent to child.

Note:

> While loading, linking & initialization if any error occurs, we will get runtime exception saying **java.lang.linkageError (java.lang.verifyError is child of linkageError).**

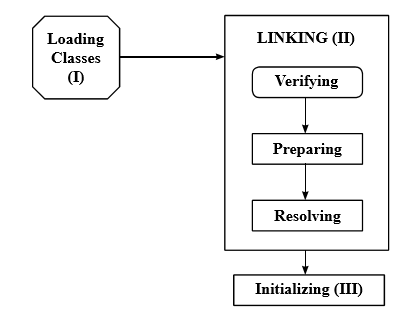


Fig: Class Loader Subsystem

Types of Class Loader Subsystem

1. **Bootstrap class loader**: - It is responsible for loading core java API classes i.e. classes present in **rt.jar** from bootstrap classpath i.e. JDK -> JRE -> lib -> rt.jar

> By default bootstrap class loader is available with every JVM.

> Not implemented in Java (implemented in native language like C/C++)

2. **Extension class loader**: - It is the child class of Bootstrap class loader. It is responsible for loading classes from extension classpath i.e. JDK -> JRE -> lib -> ext -> \*.jar

> Implemented in Java & corresponding .class file is **sun.misc.Laucher$ExtClassLoader.class**

Note: If in a class filename $ symbol is present i.e. **ExtClassLoader** is an inner class of **Laucher** outer class.

3. **Application class loader/ System class loader**: - It is the child class of Extension class loader. It is responsible to load classes from application classpath (internally uses environment variable classpath).

> Implemented in Java & corresponding .class file is **sun.misc.Laucher$AppClassLoader.class**

