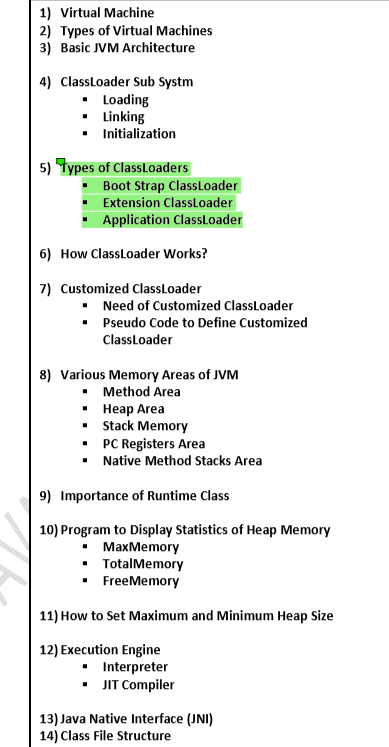
JVM Architecture



1) Virtual Machine

It is a Software Simulation of a Machine which can Perform Operations Like a Physical Machine.

**Types of Virtual Machines**

There are 2 Types of Virtual Machines

1. **Hardware Based OR System Based Virtual Machines**

**2) Software Based OR Application Based OR Process Based Virtual Machines**

**1) Hardware Based OR System Based Virtual Machines**

It Provides Several Logical Systems on the Same Computer with Strong Isolation from Each Other.

Examples:

1) KVM (Kernel Based Virtual Machine) for Linux Systems

2) VMware (Virtual Machine ware)

3) Xen

4) Cloud Computing The main advantage of Hard-ware based Virtual Machines is for effective utilization of hardware resources.

**2) Software Based OR Application Based OR Process Based Virtual Machines**

These Virtual Machines Acts as Runtime Engines to Run a Particular Programming Language Application.

Examples:

1) JVM Acts as Runtime Engine to Run Java Applications

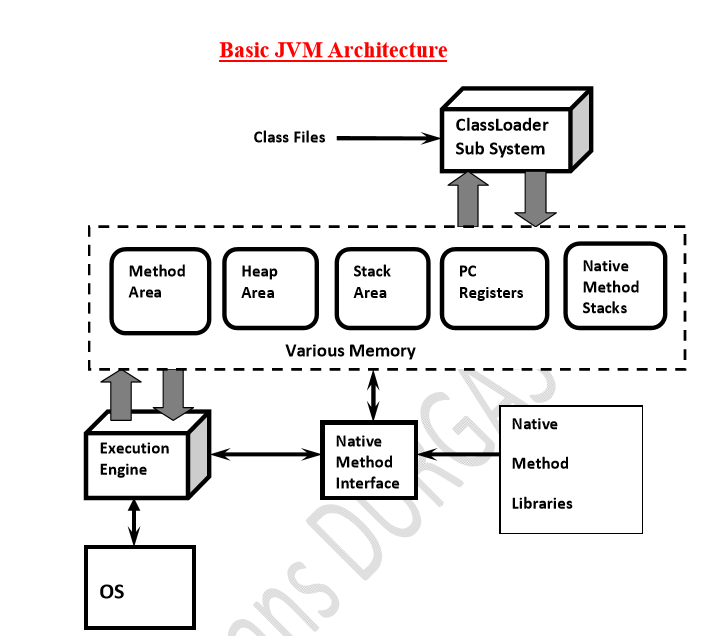
2) PVM (Parrot VM) Acts as Runtime Engine to Run Scripting Languages Like PERL.

3) CLR (Common Language Runtime) Acts as Runtime Engine to Run .Net Based Applications.

**JVM :**

JVM is the Part of JRE.

JVM is Responsible to Load and Run Java Applications



**ClassLoader Sub System:**

**ClassLoader Sub System is Responsible for the following 3 Activities.**

1. **Loading**
2. **Linking**

 **Verification**

 **Preparation**

 **Resolution**

**3. Initialization**

**1) Loading:**

 **Loading Means Reading Class Files and Store Corresponding Binary Data in Method Area.**

 **For Each Class File JVM will Store the following Information in Method Area.**

**1) Fully Qualified Name of the Loaded Class OR Interface ORenum.**

**2) Fully Qualified Name of its Immediate Parent Class.**

**3) Whether .class File is related to Class OR Interface OR enum.**

**4) The Modifiers Information**

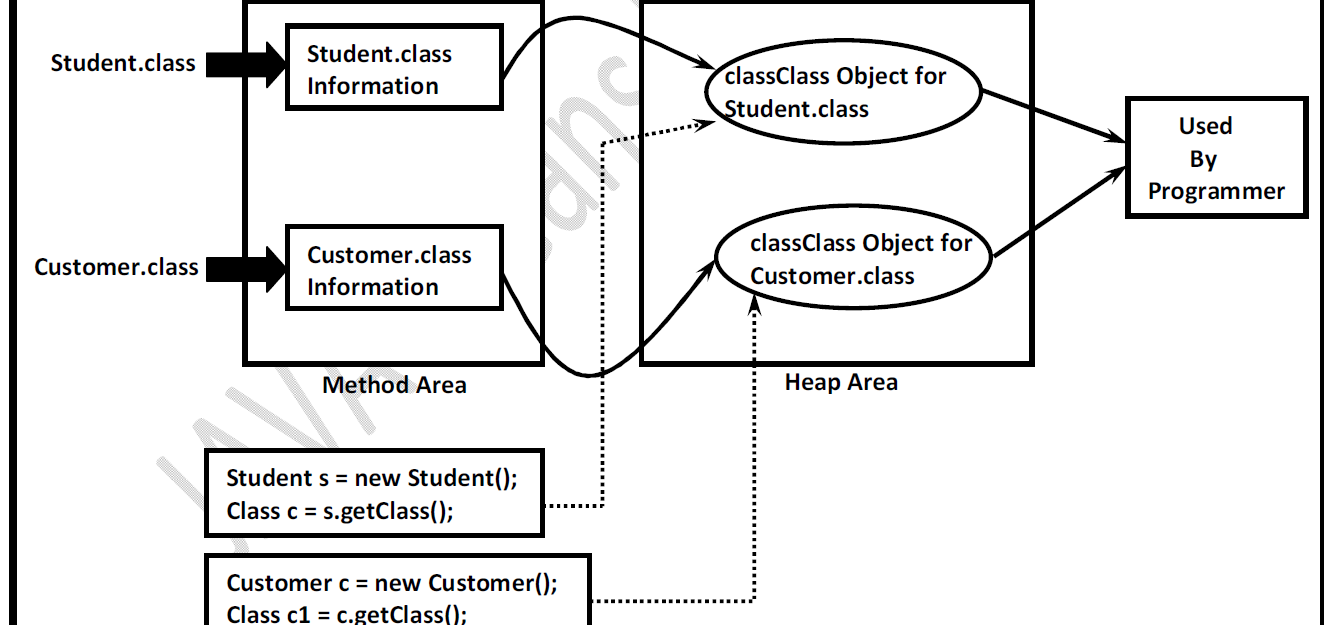
**5) Variables OR Fields Information**

**6) Methods Information**

**7) Constant Pool Information and so on.**

 **After loading .class File Immediately JVM will Creates an Object of the Type class Class to**

**Represent Class Level Binary Information on the Heap Memory.**



**The Class Object can be used by Programmer to get Class Level Information Like Fully**

**Qualified Name of the Class, Parent Name, Methods and Variables Information Etc.**

**Note: For Every loaded .class file Only One Class Object will be Created, even though we are**

**using Class Multiple Times in Our Application.**

1. **Linking:**

**Linking Consists of 3 Activities**

**1) Verification**

**2) Preparation**

**3) Resolution**

**Verification:**

 **It is the Process of ensuring that Binary Representation of a Class is Structurally Correct**

**OR Not.**

 **That is JVM will Check whether .class File generated by Valid Compiler OR Not.i.ewhether**

**.class File is Properly Formatted OR Not.**

 **Internally Byte Code Verifier which is Part of ClassLoader Sub System is Responsible for**

**this Activity.**

 **If Verification Fails then we will get Runtime Exception Saying *java.lang.VerifyError.***

**Preparation:**

**In this Phase JVM will Allocate Memory for the Class Level Static Variables and**

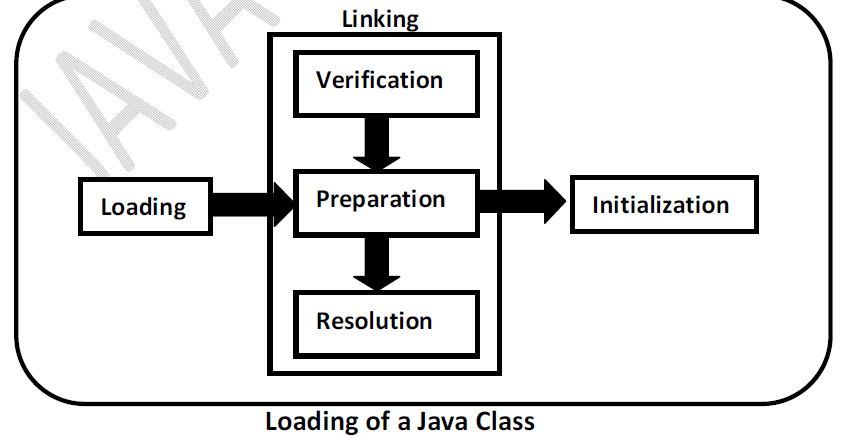
**Assign DefaultValues (But Not Original Values).**

**Note:Original Values will be assignedin Initialization Phase.**

**Initialization:**

**In this Phase All Static Variables will be assigned with Original Values and Static Blocks will**

**be executed from fromtop to bottom and from Parent to Child.**



**Types of ClassLoaders:**

**Every ClassLoader Sub System contains the following 3 ClassLoaders.**

**1) BootstrapClassLoader OR PrimordialClassLoader**

**2) ExtensionClassLoader**

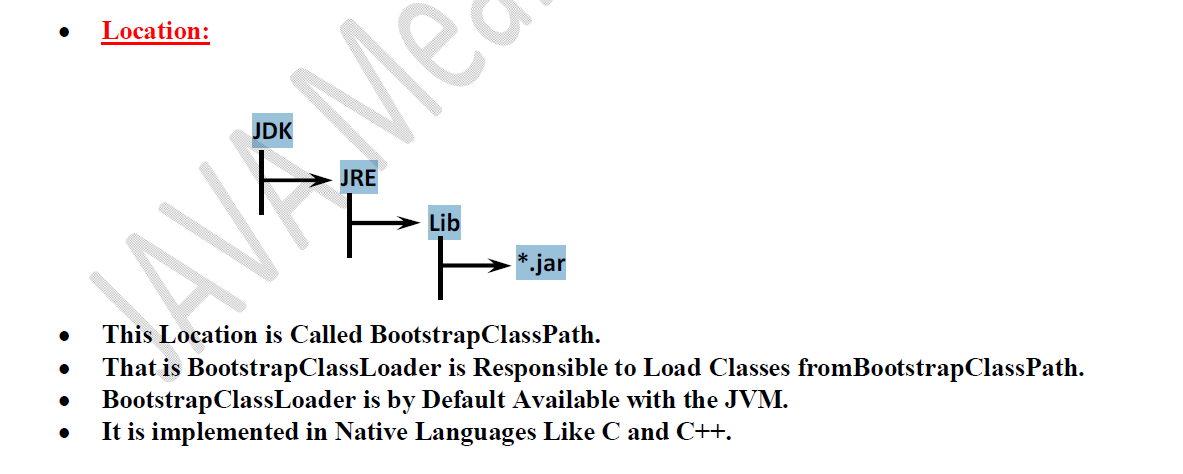
**3) ApplicationClassLoader OR SystemClassLoader**

**BootstrapClassLoader**

 **This ClassLoader is Responsible to load classes from jdk\jre\lib folder.**

 **All core java API classes present in rt.jar which is present in this location only. Hence all**

**API classes (like String, StringBufferetc) will be loaded by Bootstrap class Loader only.**

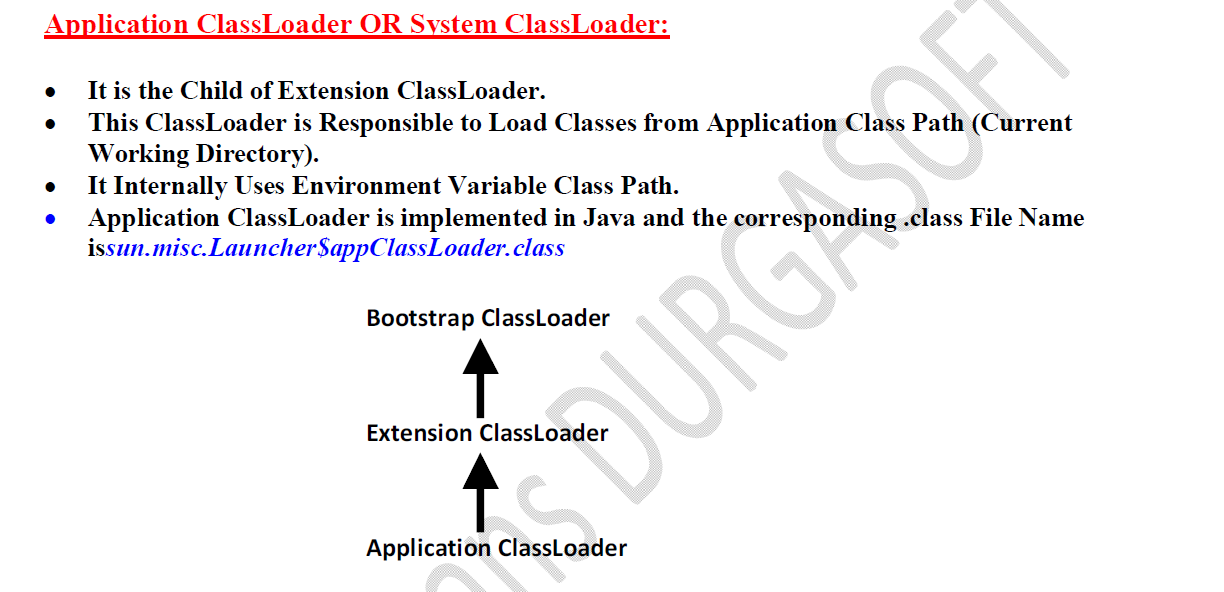


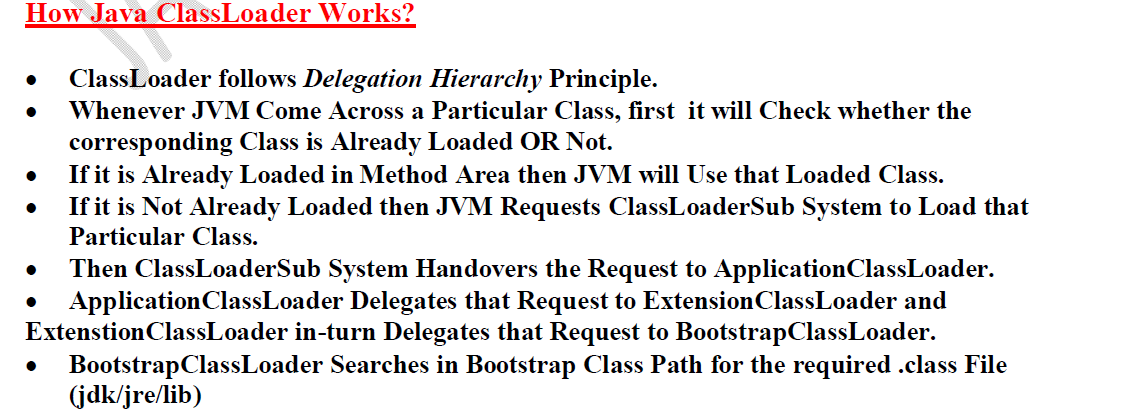
**Extension ClassLoader:**

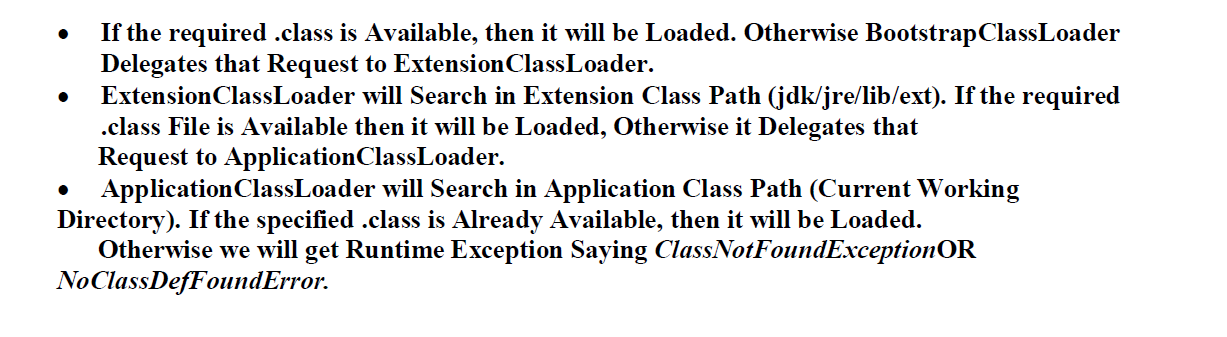
 **It is the Child of Bootstrap ClassLoader.**

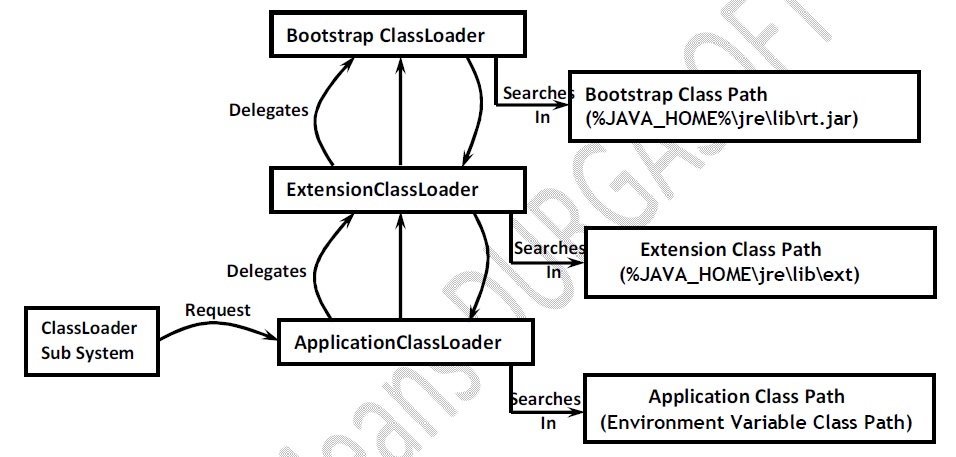
 **ThisClassLoader is Responsible to Load Classes from Extension Class Path.**

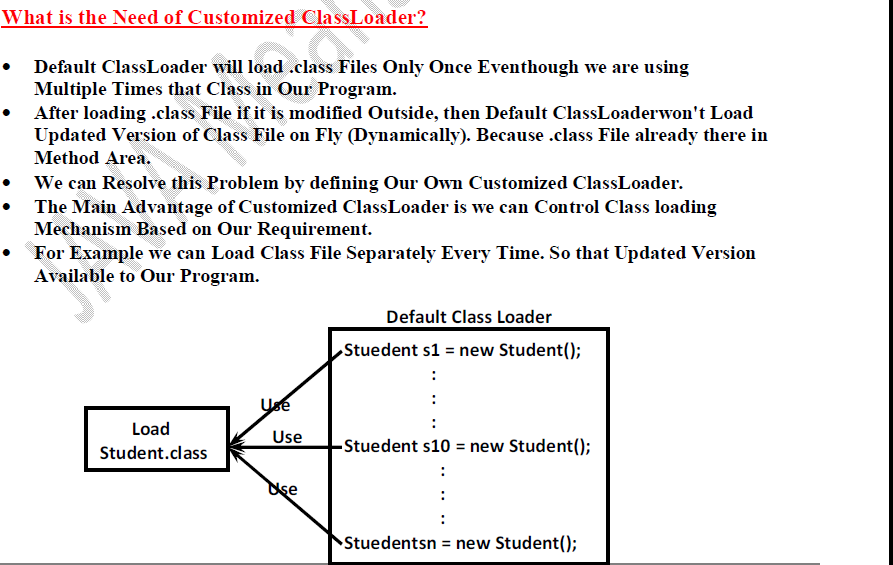
**Location:jdk\jre\lib\ext**











**What is the Purpose of java.lang.ClassLoader Class?**

 **This Class Act as Base Class for designing Our Own Customized ClassLoaders.**

 **Hence Every Customized ClassLoader Class should extends*java.lang.ClassLoader* either**

**Directly OR Indirectly.**

**Various Memory Areas of JVM:**

**Whole Loading and Running a Java Program JVM required Memory to Store Several**

**Things Like Byte Code, Objects, Variables, Etc.**

**Total JVM Memory organized in the following 5 Categories:**

1. **Method Area :**

**a) Method Area will be Created at the Time of JVM Start- Up.**

b) **It will be Shared by All Threads (Global Memory).**

c) **This Memory Area Need Not be Continuous.**

d) **Method area shows runtime constant pool.**

**2) Heap Area OR Heap Memory**

**Heap Area will be Created at the Time of JVM Start- Up.**

 **Heap Areacan be accessed by All Threads (Global OR Sharable Memory).**

 **Heap Area Nee Not be Continuous.**

 **All Objects and corresponding Instance Variables will be stored in the**

**Heap Area.**

 **Every Array in Java is an Object and Hence Arrays Also will be stored in Heap**

**Memory Only.**

**Runtime Class Present in java.lang Package and it is a Singleton Class.**

 **We can Create Runtime Object by using**

**Runtime r = Runtime.getRuntime();**

**3)Java Stacks Area:**

**For Every Thread JVM will Create a Separate Runtime Stack.**

 **Runtime Stack will be Created Automatically at the Time of Thread Creation.**

 **All Method Calls and corresponding Local Variables, Intermediate Results will be**

**stored in the Stack.**

 **For Every Method Call a Separate Entry will be Added to the Stack and that Entry is**

**Called *Stack Frame* OR *Activation Record.***

 **After completing that Method Call the corresponding Entry from the Stack will**

**beRemoved.**

 **After completing All Method Calls, Just Before terminating the Thread,the Runtime**

**Stack will be destroyed by the JVM.**

 **The Data stored in the Stack can be accessed by Only the corresponding Thread and it**

**is Not Available to Other Threads.**

**4)PC Registers Area:**

 **For Every Thread a Separate PC Register will be Created at the Time of Thread**

**Creation.**

 **PC Registers contains Address of Current executing Instruction.**

 **Once Instruction Execution Completes Automatically PC Register will be**

**incremented to Hold Address of Next Instruction.**

**5) Native Method Stacks Area:**

 **For Every Thread JVM will Create a Separate Native Method Stack.**

 **All Native Method Calls invoked by the Thread will be stored in the corresponding**

**Native Method Stack.**

**Note:**

 **Method Area, Heap Area and Stack Area are considered as *Major Memory Areas* with**

**Respect to Programmers Point of View.**

**Execution Engine:**

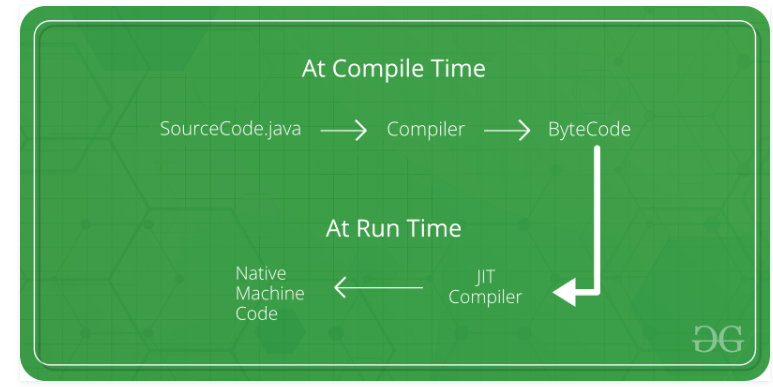
 **This is the Central Component of JVM.**

 **Execution Engine is Responsible to Execute Java Class Files.**

 **Execution Engine contains 2 Components for executing Java Classes.**

1.**Interpreter**

2. **JIT Compiler**



**Interpreter:**

 **It is Responsible to Read Byte Code and Interpret (Convert) into Machine Code (Native**

**Code) and Execute that Machine Code Line by Line.**

 **The Problem with Interpreter is it Interpreters Every Time Even the Same Method**

**Multiple Times. Which Reduces Performance of the System.**

 **To Overcome this Problem SUN People Introduced JIT Compilers in 1.1 Version.**

