**JVM (Java Virtual Machine)**

**1) Virtual Machine**

**2) Types of Virtual Machines**

**3) Basic JVM Architecture**

**4) ClassLoader Sub Systm**

 **Loading**

 **Linking**

 **Initialization**

**5) Types of ClassLoaders**

 **Boot Strap ClassLoader**

 **Extension ClassLoader**

 **Application ClassLoader**

**6) How ClassLoader Works?**

**7) Customized ClassLoader**

 **Need of Customized ClassLoader**

 **Pseudo Code to Define Customized**

**ClassLoader**

**8) Various Memory Areas of JVM**

 **Method Area**

 **Heap Area**

 **Stack Memory**

 **PC Registers Area**

 **Native Method Stacks Area**

**9) Importance of Runtime Class**

**10) Program to Display Statistics of Heap Memory**

 **MaxMemory**

 **TotalMemory**

 **FreeMemory**

**11) How to Set Maximum and Minimum Heap Size**

**12) Execution Engine**

 **Interpreter**

 **JIT Compiler**

**13) Java Native Interface (JNI)**

**14) Class File Structure**

**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**

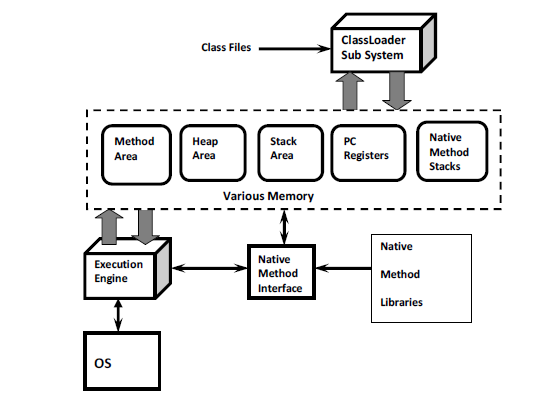
**Applications.**

**JVM**

 **JVM is the Part of JRE.**

 **JVM is Responsible to Load and Run Java Applications.**

**Basic JVM Architecture**



**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**

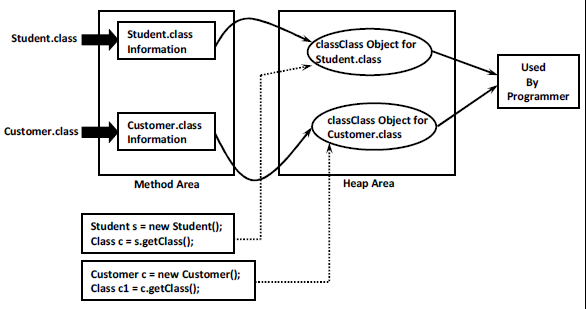
**5) Variables OR Fields Information**

**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**

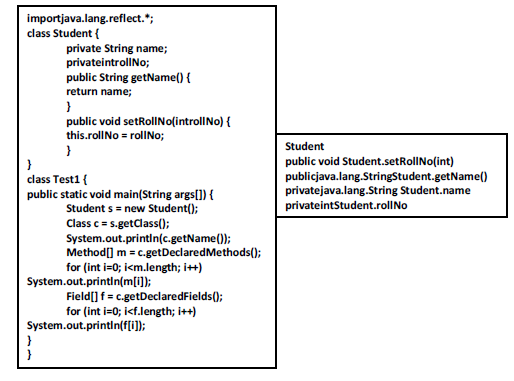
**Represent Class Level Binary Information on the Heap Memory.**



**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.**

**Program to print methods and variables information by using Class object:**

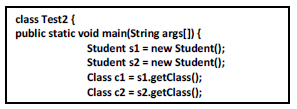


**In the Above Example by using Student class Class Object we can get its Methods and Variable**

**Information.**

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

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



**2) 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.**

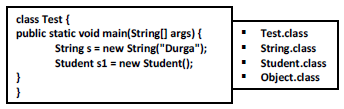
**Resolution:**

 **It is the Process of Replaced Symbolic References used by the Loaded Type with Original**

**References.**

 **Symbolic References are Resolved into Direct References by searching through Method**

**Area to Locate the Referenced Entity.**



 **For the Above Class, ClassLoadersub system Loads *Test.class, String.class,Student.class,***

**andObject.class.**

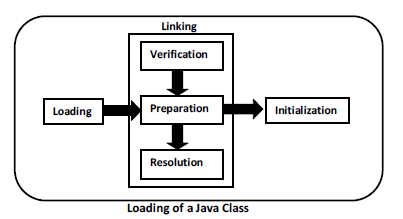
 **The Names of these Class Names are stored in *Constant Pool* of Test Class.**

 **In Resolution Phase these Names are Replaced with Actual References from Method Area.**

**3) 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.**

 **Note: While Loading, Linking and Initialization if any Error Occurs then we will get**

**Runtime Exception Saying java.lang.*LinkageError. Of course VerifyError is child class of***

***LinkageError only.***

**Types of ClassLoaders:**

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

**1) BootstrapClassLoader OR PrimordialClassLoader**

**2) ExtensionClassLoader**

**3) ApplicationClassLoader OR SystemClassLoader**

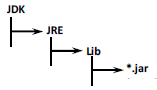
**BootstrapClassLoader**

 **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.**

 **Location:**



 **This Location is Called BootstrapClassPath.**

 **That is BootstrapClassLoader is Responsible to Load Classes fromBootstrapClassPath.**

 **BootstrapClassLoader is by Default Available with the JVM.**

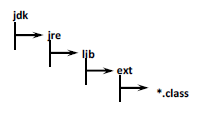
 **It is implemented in Native Languages Like C and C++.**

**Extension ClassLoader:**

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

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

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



 **This ClassLoader is implemented in Java and the corresponding .class File Name is**

***sun.misc.Launcher$ExtClassLoader.class***

**Application ClassLoader OR System ClassLoader:**

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

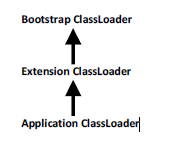
 **This ClassLoader is Responsible to Load Classes from Application Class Path (Current**

**Working Directory).**

 **It Internally Uses Environment Variable Class Path.**

 **Application ClassLoader is implemented in Java and the corresponding .class File Name**

**is*sun.misc.Launcher$appClassLoader.class***



**How Java ClassLoader Works?**

 **ClassLoader follows *Delegation Hierarchy* Principle.**

 **Whenever JVM Come Across a Particular Class, first it will Check whether the**

**corresponding Class is Already Loaded OR Not.**

 **If it is Already Loaded in Method Area then JVM will Use that Loaded Class.**

 **If it is Not Already Loaded then JVM Requests ClassLoaderSub System to Load that**

**Particular Class.**

 **Then ClassLoaderSub System Handovers the Request to ApplicationClassLoader.**

 **ApplicationClassLoader Delegates that Request to ExtensionClassLoader and**

**ExtenstionClassLoader in-turn Delegates that Request to BootstrapClassLoader.**

 **BootstrapClassLoader Searches in Bootstrap Class Path for the required .class File**

**(jdk/jre/lib)**

 **If the required .class is Available, then it will be Loaded. Otherwise BootstrapClassLoader**

**Delegates that Request to ExtensionClassLoader.**

 **ExtensionClassLoader will Search in Extension Class Path (jdk/jre/lib/ext). If the required**

**.class File is Available then it will be Loaded, Otherwise it Delegates that**

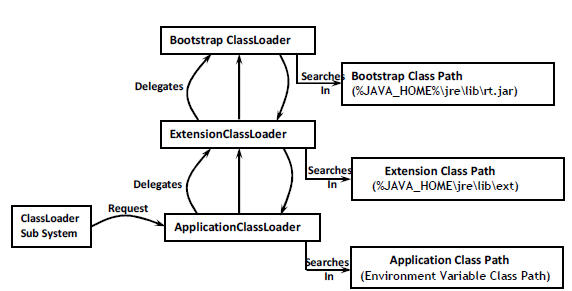
**Request to ApplicationClassLoader.**

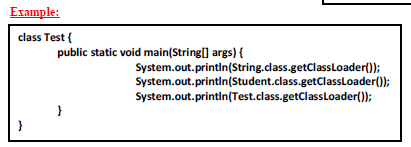
 **ApplicationClassLoader will Search in Application Class Path (Current Working**

**Directory). If the specified .class is Already Available, then it will be Loaded.**

**Otherwise we will get Runtime Exception Saying *ClassNotFoundException*OR**

***NoClassDefFoundError.***





 **For String Class: From Bootstrap Class Path by Bootstrap ClassLoader Output is null**

 **For Student Class: From Extension Class Path by Extension ClassLoader Output is**

**sun.misc.Launcher$extClassLoader@1234**

 **For Test Class:From Application Class Path by Application ClassLoader Output is**

**sun.misc.Launcher$appClassLoader@3456**

**Note: Assume that *Student.class* Present in Both *Extension Class Path* and *Application Class***

***Path* and *Test.class* Present in Only in *Application Class Path.***

**Note:**

 **Bootstrap ClassLoader is Not Java Object. Hence we are getting null in the 1st Case but**

**Extension ClassLoader and Application ClassLoader are Java Objects and Hence we get**

**Proper Output in remaining 2 Cases.**

**ClassName@HexaDecimal.String\_of\_Hashcode**

 **ClassLoader Subsystem will give Highest Priority for Bootstrap Class Path and then**

**Extension Class followed by Application Class Path.**

**What is the Need of Customized ClassLoader?**

 **Default ClassLoader will load .class Files Only Once Eventhough we are using**

**Multiple Times that Class in Our Program.**

 **After loading .class File if it is modified Outside, then Default ClassLoaderwon't Load**

**Updated Version of Class File on Fly (Dynamically). Because .class File already there in**

**Method Area.**

 **We can Resolve this Problem by defining Our Own Customized ClassLoader.**

 **The Main Advantage of Customized ClassLoader is we can Control Class loading**

**Mechanism Based on Our Requirement.**

 **For Example we can Load Class File Separately Every Time. So that Updated Version**

**Available to Our Program.**