**JVM**

**1.Virtual :-** The real-world is physical. You,I,our relatives and your friends in this real-world are physical. Our dream world is virtual. I ,you,our relative and your friends in dream world are virtual.

**Virutal is not physical. It is logical representation of physical thing.**

In computer world the meaing of virtual is:

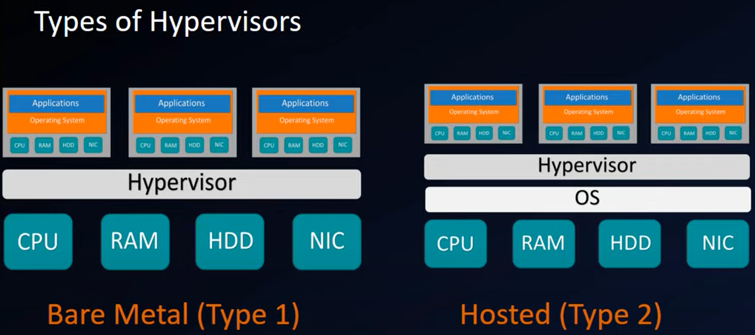
A logical representation of physical thing created in a computer through software.

Ex:

1. In bike Racin game , the bike is virtual bike.
2. In Temple Run game, the person who jumping and running is virtual person.

**2. Virtual Machine:-** The virutal Machine is virtual instance of computer . That performs almost of all functions as real computer.

2.1.Hypervisor:- A**hypervisor**, also known as a **virtual machine monitor** or VMM, is software that creates and runs virtual machines (VMs). There are two types of Hypervisor.



1.Bare Metal :- This hypervisor can act also as Operating system. It can be directly hosted on hard-ware with out operating system.

2. Hosted:- This hypervisor should be instally on OS.

Examples to Hypervisor: Microsoft Hyper-v, Vmware,Redhat virtulization…etc.

Benefit:

1. In single computer, we can run multiple os on single piece of hardware.

Ex: we can install windows xP/Linux /…etc in single computer with help of virtual Machine.

**3. JVM(Java virtual Machine):-**

**Def1:** The JVM is specification. That means it is abstract documentation which provides rules and guide line for implementing JVM functionality.

**Def2:** JVM is software implementation of JVM specification. The specification is implemented by different vendors in world such as SUN,ORACLE,IBM,..etc.

**Def3**: JVM is runtime instance means when we run java command one separate instance of JVM is created for running the given java application.

By using “java” tool, we initiate the JVM execution.

Syntax: java .classfilename

When we run java command, one instance of JVM is created as process in OS and it occupies some space in RAM.

The JVM provides Run time environment for executing the Java application.

Examples to JVM Based Languages are “**java, groovy, koltin and scala”**.

**3.1 .Types of JVMS**:- The Java SE contains two implementations of JVM.

a. Java HotSpot Client VM

b. Java HotSpot Server VM.

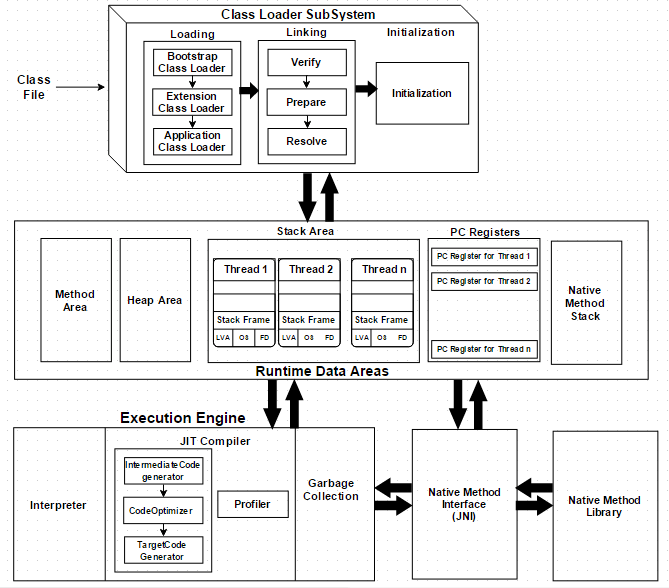
##### a.Java HotSpot Client VM:- It is default VM. By default java program is executed with client VM. The Java HotSpot Client VM has been specially tuned to reduce application start-up time and memory footprint, making it particulary well suited for client environments. It is available Java2 onwards.

* Java abc 🡪 By default abc is executed by client VM.

**b.Java HotSpot Server VM:-**  The java HotSPot server VM is designed for maximum program execution speed for applications running in server enviornment.

> java –server Test 🡪 class Test is executed by ServerVM.

**4. JVM Architecture:-**



The JVM have four components:

1.Class Loader SubSystem.

2. Runtime Data Areas.

3. Execution Engine

4. Java Native Interface(JNI).

**5.Class Loader SubSystem:-** The class loader loads and stores given class byte code in JVM method area.

**5.1.Loading Phase:-**It consists of 3 sub loaders for loading the JAVA API classes and user defined classes.

i. BootStrap class Loader.

ii. Extension class Loader.

iii. Application class Loader.

* Java .classfilename ->

Step1:JVM first checks for class byte code in method area. If it is already loaded, it makes use of loaded class byte code type. If the class is not yet loaded, JVM requests the class loader subsystem to load this class.

Step 2: The class loader sub system handovers request to Application Class Loader. The application class loader handover this request to Extension Class loader without searching this class in folders configured in “Classpath environment variable”.

Step 3: The Extension again handover this request to Bootstrap class loader. Now Bootstrap class loader search folders configured in bootstrap Class path.

Step 4: If class is found, It loads class into JVM method area. Otherwise It handover request back to Extension class loader.

Step 5: Extension class loader search folders configured in Extension class path. If class is found, It loads class into JVM method area. Otherwise It hand over request back to Application class loader.

Step 6: Application class loader search folders configured in Extension class path. If class is found, It loads class into JVM method area. Otherwise

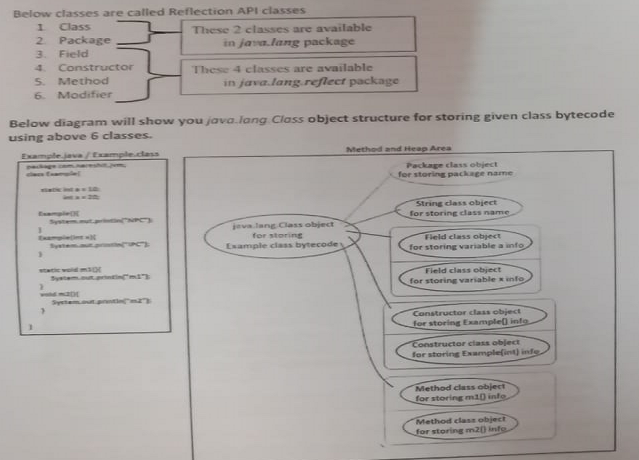
We get one of two error.

Java.lang.ClassNotFoundException.

Java.lang.NoClassDefFoundError.

Note: If class is found in any one of the classpaths, the respective classLoader will load this class into JVM’s Method area by creating java.lang.class instance.

**This java.lang.Class** instance will contains the byte code of loaded class. The java.lang.Class object internally uses several other classes objects for storing given class byte code. The classes which are used for storing given class byte code in JVM are collectively called **“Reflection API”.**



**5.2.Linking Phase: The**  following tasks are done in this phase.

* *Verification*: It ensures the correctness of the *.class* file i.e. it checks whether this file is properly formatted and generated by a valid compiler and contains any virus or not. If verification fails, we get run-time exception ***java.lang.VerifyError***. This activity is done by the component ByteCodeVerifier. Once this activity is completed then the class file is ready for compilation.
* *Preparation*: JVM allocates memory for class static variables and initializing the memory to default values.
* *Resolution*: It is the process of replacing symbolic references from the type with direct references. It is done by searching into the method area to locate the referenced entity.

**5.3 Initialization Phase:-**  The static block is executed and static variables are initialized with their initial values given in their declaration.

After initialization phase, The execution of main method is started.

**6. Run Time Data Areas:-** JVM totally contains five runtime areas

1. Method Area

2. Heap Area

3. Java Stacks Area

4. Program counter Registers Area

5. Nativ Methods stacks area.

6.1. Method Area:- **This java.lang.Class** instance will contains the byte code of loaded class.This instance is stored in method area. It means the static variables ,staticblocks logic, instance varibales, blocks logic instance methods logic , and constructors logic is stored in java.lang.Class instance which is in method area. As many classes are loading into JVM , those many java.lang.Class instances are created in Method area.

Static variables memory is allocated in this memory area.

6.2. Java Stacks Area:- By default ,JVM creates two threads in Java Stacks area.

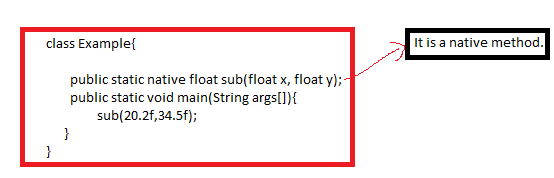
1. Main thread:When method is invoked/constructors/static blocks/blocks , new stack frame is created in thread stack. Static block/block/constructor/metods logic is loaded in this stack frame from method area. The parameters,local variables of static block/non-static block/methods/constructors gets memory in this frame. Once this method execution completed stack frame is destroyed.

2. garbage collector thread:- It is responsible to destroy the unused object from heap area.

Note:- Inside a thread if there is no sufficient memory for creating new stack frame for execution a method, then JVM will throw exceptin **java.lang.StackOverflowError**.

6.3. Program Counter Registers Area:- The PC register is created when new thread is created; after thread execution is completed it is destoryed automatically. This register stores the current executing instruction address.

6.4. Native Methods Stacks Area:-



In Java , to create native method, we must place native keyword in its prototype and it should not have body.

The native method definition is written in some other language such as c,c++ or assembly.

The above code compiled successfully. But in execution JVM will throw exception: **java.lang.UnsatisfiedLinkError.**because we just defined native method, but we did not defined it required C program and not linked.

By using JNI-Java Native interface, it is mediator between java native method and c/c++ method definiton. It provides linking bettween Java native method and c/c++ method definiton. It executes the c/C++ method and returns the result to java program.

These native methods logic is stored and executed in **Native method stack area**.

6.5.Heap Area:- All objects are created in heap area.

**7.Execution Engine:-** It is main component in JVM. It communicates with various memory areas of JVM. It is responsible for executing a java byte code. It contains mainly 3 components.

a. interpreter

b.JIT

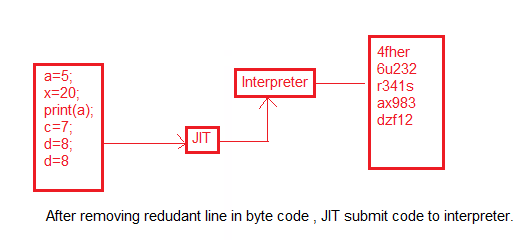
c. Garbage collector.

7.1. Interpreter:- It reads the byte code and [interprets](https://www.geeksforgeeks.org/compiler-vs-interpreter-2/)(convert) into the machine code(native code) and executes them in a sequential manner. This component runs the application from the command line by accepting a filename argument. The problem with the interpreter is that it interprets every time, even the same method called multiple times, which reduces the performance of the system. To overcome this problem JIT Compilers introduced.

7.2 JIT(Just in time compiler):- The invocation of compiler is happened just before the interpretation. JIT analyzes the byte code and remove the redundancy so that interpretation happens very speedly.

JIT compiler will find the code which is executed repeatedly, generates machine language code for this method only once, buffer and reuses the sam machine language code for repeated execution. This is how Jit compiler will give high performance in execution.

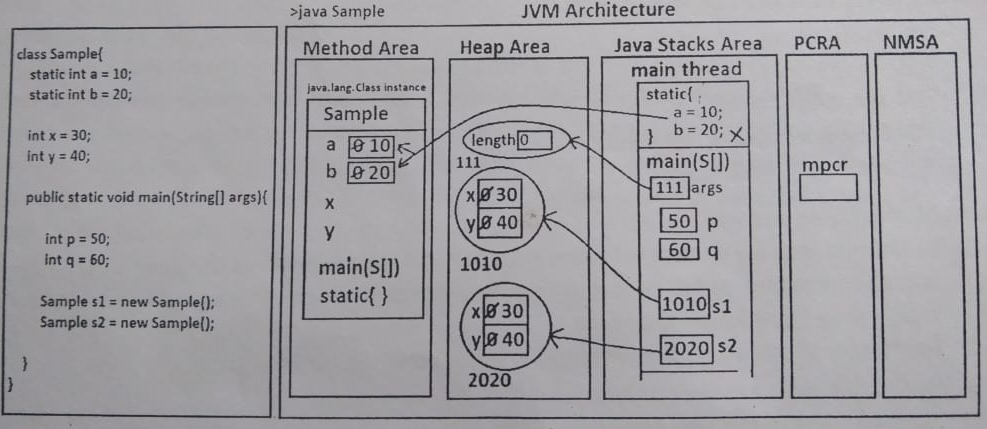
Example:



7.3 Garbage Collector:- It is daemon thread which always runs in the back ground. It remove unused objects and unreferenced objects from heap area.

**Example**

Understand all above points practicaly with program execution with all four types of variable and their meory distribution done inside runtime data areas.



Step1. When we run above class Sample, the java launcher, will create JVM as process.

Step 2. Once JVM set up is completed, JVM will search for the sample class byte code in Method aea. Since sample class byte cod is not found, JVM will request class loader sybsystem to load the Sample class byte code into JVM.

Step 3. AppClassLoader will load and store sample class byte code inside method area by creating java.loang.Class object.

Step4. After butecode verification is completed , in preparation phase memory will be allocated for static variables a and b in their class context in Method area with default values zero.

Step 5. Then in intialization phase, static block is executed by creating a stack frame inside main thread in java Stack Area, static variables are initialzied with assigned values 10 and 20. Once after static block execution is completed stack frame is destroyed.

Step 6. After initializaiton, in execution phase, main method logic is executed by creating separate stack frame in main thread in java Stack Area. Inside main method stack frame its parameter args, local variables p,q, s1 and s2 memory is allocated.

Step 7. In heap area one empty String[] array object, and two objects of Sample class are created. Non-static variables x, y memory is allocated in heap area in their objects. Since we have created two objects from Saple class, two copies of non-static variables memory is created.

Step8. Once main method execution completed, its stac frame is destroyed, main thread is destroyed and saple class is unloaded, JVM destroyed.

Step9. In Pc regiesters area one pc register is created for tracking main trherad execution.

Step10. This complete executin is tracked by execution engine one of the JVM components.