**JVM (Java Virtual Machine)** is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.  
JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.  
The JVM performs following main tasks:  
**Loads code**  
**Verifies code**  
**Executes code**  
**Provides runtime environment**

**JRE** is an acronym for Java Runtime Environment. It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.  
Implementation of JVMs are also actively released by other companies besides Sun Micro Systems.

**JDK** is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools.

JVM Architecture

* This topic clears your questions like Explain jvm architecture in java, jvm architecture in java with diagram, how JVM works internally and areas of Java Virtual Machine.
  + **Virtual machine:** In general terms Virtual machine is a software that creates an environment between the computer and end user in which end user can operate programs.
  + As per functionality Virtual machine is creation of number of different identical execution environments on a single computer to execute programs is called Virtual machine.
  + **Java Virtual machine:** it is also Virtual machine that runs java bytecode by creating five identical run time areas to execute class members.

**Function of java virtual machine**

* The bytecode is generated by java compiler in a JVM understandable format.
* As a programmer we develop a java application and when we compile a java program, the compiler will generate .class (dot class) file.
* The .class file contains byte code (Special java instructions).
* To execute a java program we take the help of JVM (java virtual machine) to the JVM we have to provide .class file as the input.

**Types of JVMs:**

* The java 2 SDK , Standard Edition , contains two implementation of the java virtual machine

1. Java HotSpot Client VM
2. Java Hotspot Server VM

**Java HotSpot Client VM:**

* The Hotspot Client VM is the default virtual machine of the java 2 SDK and java 2 run time environment .
* As its name implies, it is tuned for best performance when running applications in a client environment by reducing application start up time and memory footprint.

**Java HotSpot Server VM:**

* The java HotSpot server VM is designed for maximum program execution speed for applications running in a server environment.
* The Java HotSpot Server VM is invoked by using the server command line option when an application , as in java server MyApp.
* Some of the features java HotSpot technology , common to both VM implementations, are the following.

**Adaptive compiler:**

* Applications are launched using a standard interpreter, but the code is then analyzed as it runes to detect performance bottlenecks, or "hot spots".
* The Java HotSpot VMs compile those performance critical portions of the code for a boost in performance , while avoiding unnecessary compilation of seldom used code.
* The Java HotSpot VMs also uses the adoptive compiler to decide on the fly , how best to optimize compiled code with techniques such as in lining.
* The run time analysis performed by the compiler allows it to eliminate guesswork in determining which optimizations will yield the largest performance benefit.

**Rapid memory allocation garbage collection:**

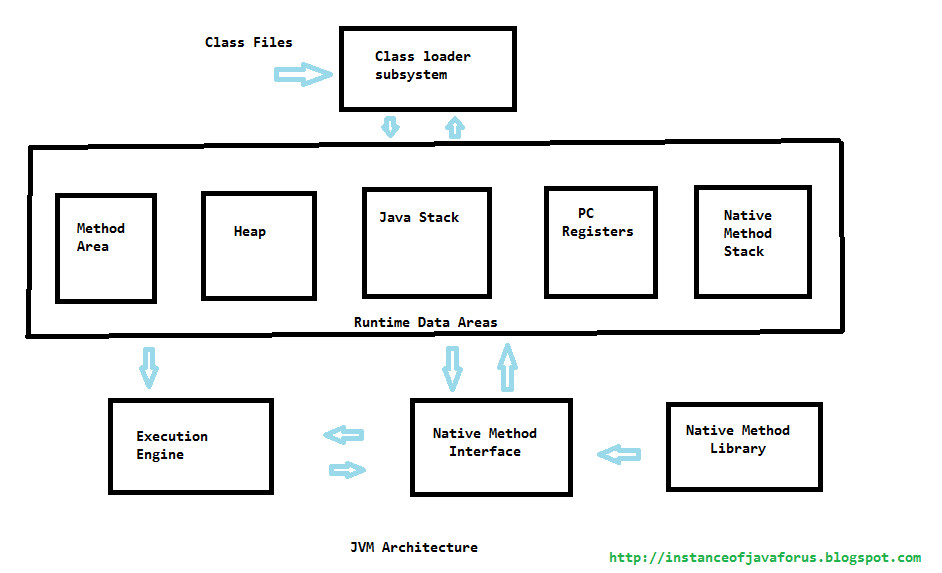
* Java HotSpot technology provides for rapid memory allocation for objects , and it has a fast, efficient, state-of-the-art  garbage collector.

**Thread Synchronization:**

* The Java programming language allows for use of multiple , concurrent paths of program execution (called threads).
* Java HotSpot technology provides a thread- handling capability that is designed to scale readily for use in large , shared memory multiprocessor servers.

**Class Loader Sub System:**

* The class loader sub system will take a .class file as the input and performance the following operations.
* The class loader sub system is responsible for loading the .class file (Byte code) into the JVM.
* Before loading the Byte code into the JVM it will verify where there the Byte code is valid      or   not. This verification will be done by Byte code verifier.
* If the Byte code is valid then the memory for the Byte code will be allocated in different areas.
* The different areas into which the Byte code is loaded are called as Run Time Data Areas. The various run time data areas are.



**1.Method Area:**

* Java Virtual Machine Method Area can be used for storing all the class code and method code.
* All classes bytecode is loaded and stored in this run time area , and all static variables are created in this area.

**2.Heap Memory:**

* JVM Heap Area can be used for storing all the objects that are created.
* It is the main memory of JVM , all objects of classes :- non static variables memory are created in this run time area.
* This runtime  area memory is finite memory.
* This area can be configured at the time of setting up of runtime environment using non standard option like
* java -xms <size> class-name.
* This can be expandable by its own , depending on the object creation.
* Method area and Heap area both are sharable memory areas.

**3.Java Stack area:**

* JVM stack Area can be used for storing the information of the methods. That is under execution. The java stack can be considered as the combination of stack frames where every frame will contain the stat of a single method.
* In this runtime area all Java methods are executed.
* In this runtime JVM by default creates two threads. they are
* 1.main method
* 2.Garbage Collection Thread
* main thread responsible to execute java methods starts with main method.
* Also responsible to create objects in heap area if its finds **new** keyword in any method logic.
* Garbage collection thread is responsible to destroy all unused objects from heap area.

**4.PC Register (program counter) area:**

* The PC Register i Java Virtual Machine will contain address of the next instruction that have to be executed.

**5.Java Native Stack:**

* Java native stack area is used for storing non-java coding available in the application. The non-java code is called as native code.

**Execution Engine:**

* The Execution Engine of JVM is Responsible for executing the program and it contains two parts.

1. Interpreter.
2. JIT Compiler (just in time compiler).

* The java code will be executed by both interpreter and JIT compiler simultaneously which will reduce the execution time and them by providing high performance. The code that is executed by JIT compiler is called as HOTSPOTS (company).