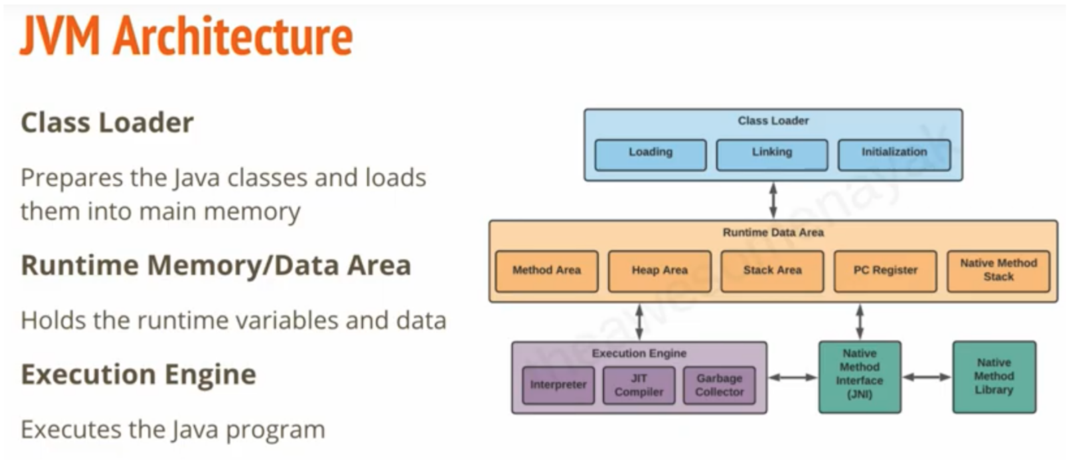
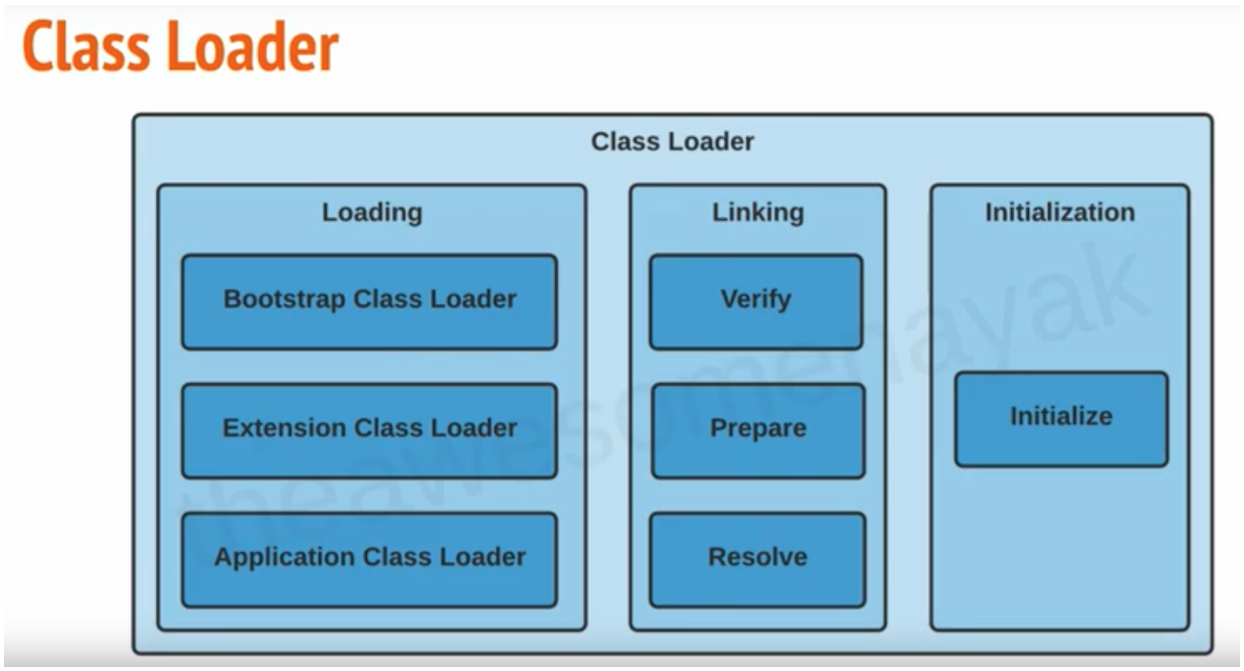
1. **What is JVM -**



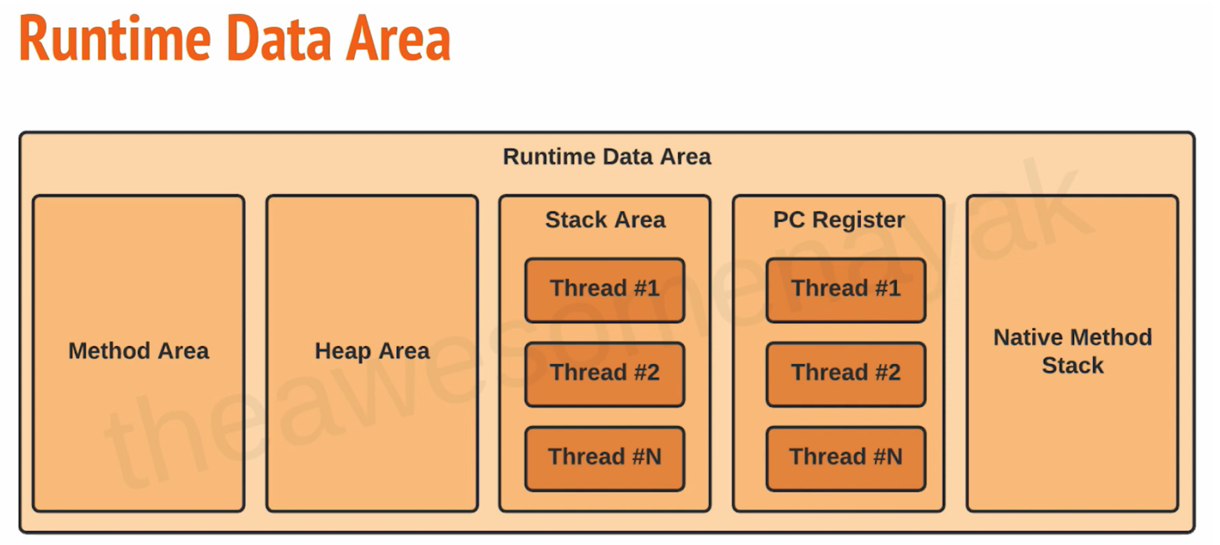
The JVM is a virtual machine that runs Java class files in a portable way. Being a virtual machine means that JVM is an abstraction of an underlying, actual machine – such as the server that your program is running on. Regardless of what operating system or hardware is actually present, the JVM creates a predictable environment for programs to run within. Unlike a true virtual machine, however, the JVM doesn't create a virtual operating system. It would be more accurate to describe the JVM as a managed runtime environment, or a process virtual machine.



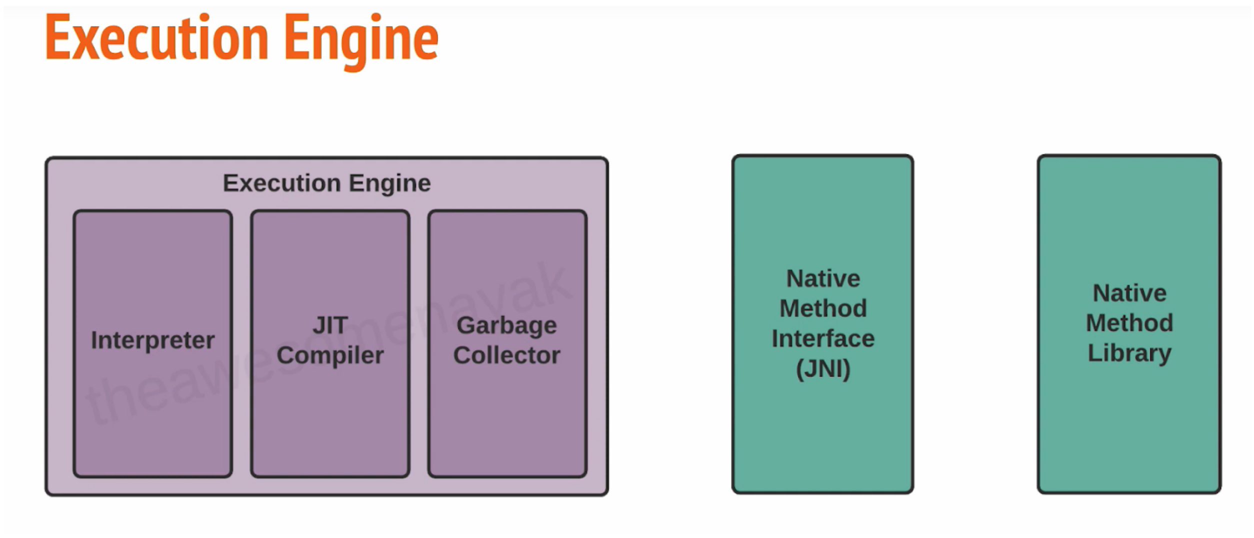
Class Loader -

1) Bootstrap ClassLoader - this is the first classloader which is the super class of Extension classloader. It loads the rt.jar file which contains all class files of Java Standarrd Edition like java.lang, java, net package classes. Java.util package classes, java.io package classes, java.sql package classes etc.

2) Extension classLoader: child classloader of Bootstrap and parent classloader of System classloader. It loads the jar files located inside $JAVA_HOME/jre/lib/ext directory.
3) System/Application ClassLoader: This is the child classloader of the Extension classloader. It loads the class files from classpath. By default, the classpath is set to the current directory. You can change the classpath using the “-cp” or “-classpath” switch. It is also known as the Application classloader.



Runtime Data Area -
1) Class(Method) area - stores per-class structures at the runtime constant pool, field and method data, the code for methods.
2) Heap - It is the runtime data area in which objects are allocated
3) Stack - stores frames. Each thread has a private JVM stack, created at the same time as the thread.
4) PC register contains the address of the Java virtual machine instruction currently being executed.
5) Native method Stack - it contains all the native methods used in the application.



Extension Engine -

1) Interpreter - The interpreters the bytecode faster but execute slowly. The disadvantage of the interpreter is that when one method is called multiple times, every time a new interpretation is required.
2) JIT Compiler - The JIT Compiler neutralizes the disadvantage of the interpreter. The execution engine will be using the help of the interpreter in converting byte code, but when it finds repeated code it used the JIT compiler, which compiles the entire bytecode and changes it to native code. This native code will be used directly for repeated method calls, which improve the performance of the system.
3) Garbage Collector: Collects and removes unreferenced objects. Garbage Collection can be triggered by calling System.gc(), but the execution is not guaranteed. Garbage collection of the JVM collects the objects that are created.

Java Native Interface (JNI): JNI will be interacting with the Native Method Libraries and provides the Native Libraries required for the Execution Engine.

Native Method Libraries: This is a collection of the Native Libraries, which is required for the Execution Engine.

References - https://www.javatpoint.com/jvm-java-virtual-machine
https://www.infoworld.com/article/3272244/what-is-the-jvm-introducing-the-java-virtual-machine.html
https://docs.google.com/document/d/1FFk9Uxcoc603C1zHSPdGM8M5uBepfD5VQyx7GmEsKNc/edit
https://dzone.com/articles/jvm-architecture-explained