**BASICS OF JAVA**

Java is a famous language which is secure, high-performance, robust, portable and platform- independent.

Platform Independence: Works on the principle of “**Write once and run anywhere**”.

This platform independence is because of JVM.

Java is platform-independent because it is compiled to a byte-code that can be run on any device that has a Java Virtual Machine (JVM). This means that you can write a Java program on one platform (such as Windows) and then run it on a different platform (such as macOS or Linux) without making any changes to the code.

In the Java programming language, a keyword is any one of 68 reserved words that have a predefined meaning in the language.

Public in Java is that it can be accessed everywhere.

Private: for that respective scope or class only.

In java the execution starts from the ‘main’ only.

“**public static void** **main(String args[])**” entry point for the execution.

**“System. out. println()”** for printing anything.

Format to save the file is “class\_name.java”

public class Test {

public static void main () {}

}

So, we’ll be saving the above file with the name of “Test.java”.

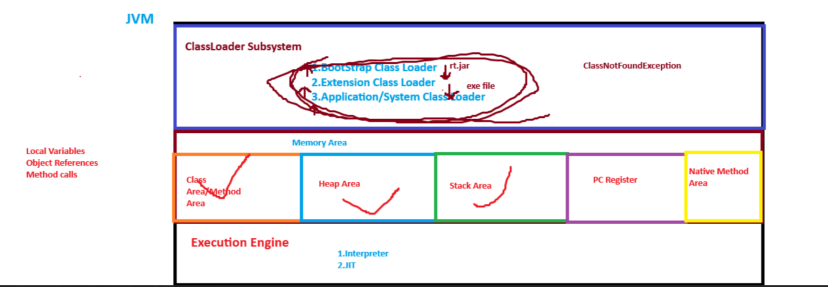
Now we’ll saved it using the class\_name.java

After this for the compilation we’ll use “javac file\_name.java” or “javac class\_name.java”. And in our case it will be “javac Test.java”.

Under this java compiler checks line by line whether whatever has been written is in java syntax or not.

After this compilation javac Test.java one file will be created with the name of Test (which is a class file). This class file has binary data in terms of 0’s and 1’s.

For the running of the program we’ll use java file\_name or java class\_name and in our case, it’ll be “java Test”.



**ClassLoader Subsystem (Inside the JVM): Helps us to load the (.class file).**

**Object is the super class for all the classes in java.**

The flow inside JVM starts like the flow will start from Application/System class loader and then to Extension Class Loader and then to Bootstrap Class Loader and then the class file will be checked by Bootstrap Class Loader whether its having rt.jar or not or whether this rt.jar is present inside the Test.java or not and if it’s present Bootstrap Class Loader will load it.

rt.jar file belongs to the Bootstrap Class Loader.

* **ClassNotFoundException:**

If rt.jar isn’t present in Bootstrap Class Loader, we’ll check in the Extension Class Loader to check for the exe file and if not found will go to the Application/System Class Loader and if not found in all these 3 Class Loaders then we’ll be getting this error which is **ClassNotFoundException**.