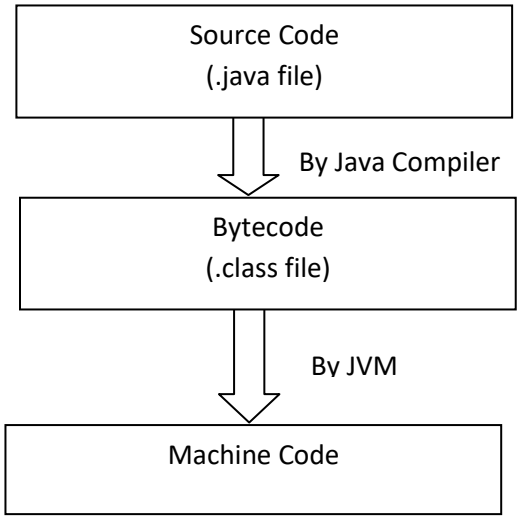
**Java Program Execution:**

Any java program execution has the following set of steps:

1) Whatever code we write using IDE, Notepad, etc, it is called as the source code(.java file).
2) This source code is converted to bytecode(.class file) using compiler. This bytecode is machine-independent and hence makes java language as platform independent.
3) Further this bytecode is converted to machine code by the JVM which is custom-built for every operating system. . This machine code is machine/OS specific.




**4)** Due to the two-step execution process described above, a java program is independent of the target operating system. However, because of the same, the execution time is way more than a similar program written in a compiled platform-dependent program.

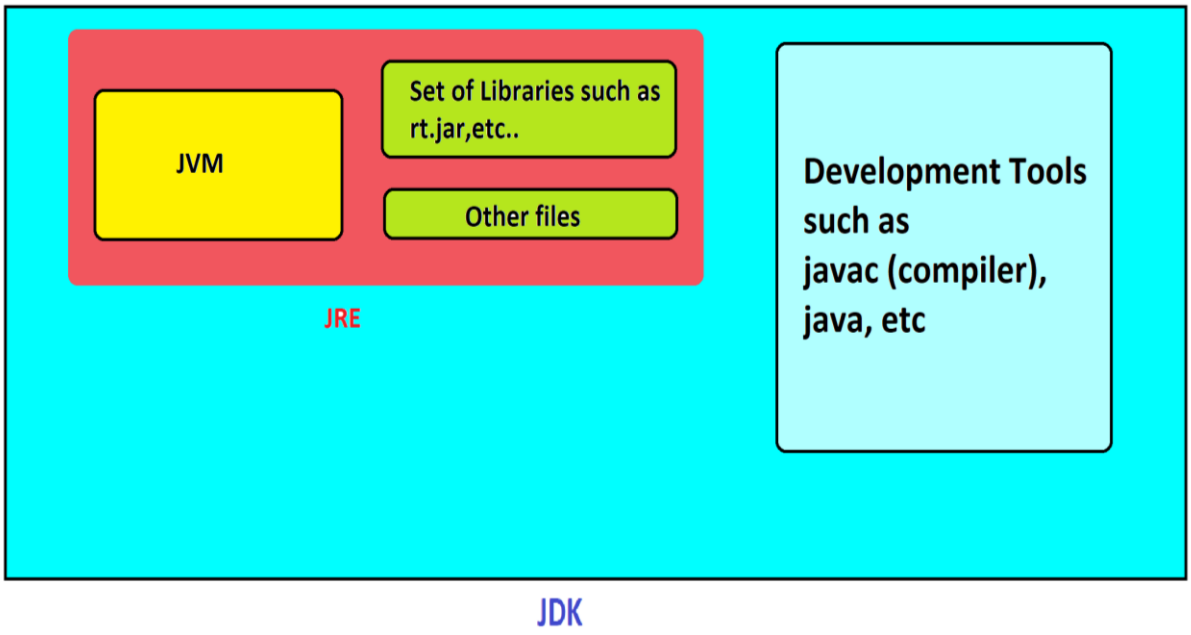
## JVM JRE and JDK :

## JDK:

**Java Development Kit (JDK)** is a software development environment used for developing Java applications. It includes the Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and other tools needed in Java development.

## JRE:

JRE stands for **Java Runtime Environment**. The Java Runtime Environment provides the minimum requirements for executing a Java application; it consists of the Java Virtual Machine (JVM), core classes, and supporting files like rt.jar files.




## JVM:

JVM stands for **Java Virtual Machine**. It is a part of JRE. JVM is responsible to load and run the java program.

JVM Runs Java Byte Code by creating 5 Identical Runtime Areas to execute Class Members.

- Class Loader Sub System
- Memory Management System

• PC-Registers
• Execution Engine
• Native Methods Stack

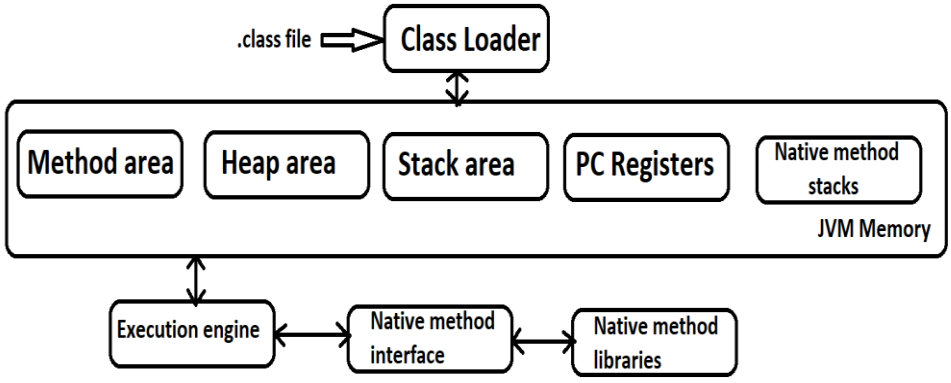



Fig. JVM Arch

**a) Class Loader:**

Class Loader is used to load the .class file in the memory.

**b) Memory Management System:**

While Loading and Running a Java Program JVM required Memory to Store Several Things Like Byte Code, Objects, Variables, Etc.

Total JVM Memory organized in the following 5 Categories:

- Method Area
- Heap Area OR Heap Memory
- Java Stack Area
- PC Registers Area
- Native Method Stacks Area

**Method Area:**

- Total Class Level Binary Information including Static Variables Stored in Method Area.

**Heap Area:**

- All Objects and corresponding Instance Variables will be stored in the Heap Area.

**Java Stack Area:**

• All Method Calls and corresponding Local Variables, Intermediate Results will be stored in the Stack.

**c) PC Registers Area**

• For Every Thread a Separate PC Register will be created at the Time of Thread Creation.
• PC Registers contains Address of Current executing Instruction.
• Once Instruction Execution Completes Automatically PC Register will be incremented to Hold Address of Next Instruction.

**d) Execution Engine**

This is the Central Component of JVM.
• Execution Engine is Responsible to Execute Java Class Files.
• It contain information about JIT (Just in time) compiler and interpreter

**e) Native Methods Stack**

• For Every Thread JVM will Create a Separate Native Method Stack.
• All Native Method Calls invoked by the Thread will be stored in the corresponding Native Method Stack

**f) Java Native Interface (JNI):**

JNI Acts as Bridge (Mediator) between Java Method Calls and corresponding Native Libraries.

**g) Java Native Library:**

Java Native Library is the collection of Native methods which are required in java.
Native method is a method declared in java, but, implemented in non java programming languages like C , C++,...