What is JVM

Introduction :

- JVM acts as a run-time engine to run Java applications.
- JVM is the one that actually calls the main method present in a java code. JVM is a part of JRE Java Runtime Environment.
- Java applications are called WORA Write Once Run Anywhere. This means we can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustment.
- This is all possible because of JVM.

- JVM Memory
- Method area :
- In the method area, all class level information like class name, immediate parent class name, methods and variables information etc. are stored, including static variables.
- There is only one method area per JVM, and it is a shared resource. From java 8, static variables are now stored in Heap area.
- Heap area:
- Information of all objects is stored in the heap area. There is also one Heap Area per JVM.

- Stack area:
- For every thread, JVM creates one run-time stack which is stored here.
- Every block of this stack is called activation record/ stack frame which stores methods calls. All local variables of that method are stored in their corresponding frame.
- After a thread terminates, its run-time stack will be destroyed by JVM.
- It is not a shared resource.

- **PC Registers :**
- Store address of current execution instruction of a thread. Obviously, each thread has separate PC Registers.
- **Native method stacks:**
- For every thread, a separate native stack is created. It stores native method information.

- **Execution Engine :**
- Execution engine executes the bytecode.
- It reads the byte-code line by line, uses data and information present in various memory area and executes instructions.
- **Interpreter:**
- It interprets the bytecode line by line and then executes.
- The disadvantage here is that when one method is called multiple times, every time interpretation is required.

- **Just-In-Time Compiler(JIT) :**
- It is used to increase the efficiency of an interpreter.
- It compiles the entire bytecode and changes it to native code so whenever the interpreter sees repeated method calls, JIT provides direct native code for that part so re-interpretation is not required, thus efficiency is improved.
- **Garbage Collector:**
- It destroys un-referenced objects.