Lesson:

JVM, Variables in Java

List of Concepts Involved:

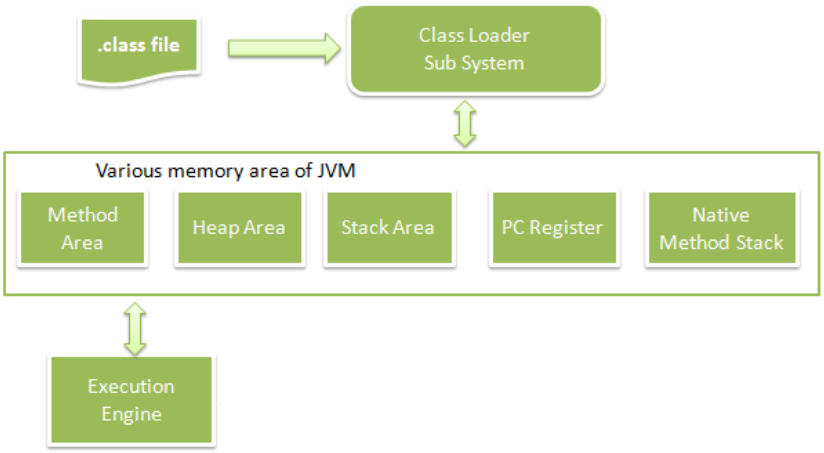
- JVM Area and its Architecture
- Types of variables and its scope
- var-args in java
- new vs newInstance(), instanceof vs isInstance()

Architecture Diagram Of JVM

In Java, JVM is part of JRE and it is responsible for loading and running JAVA class files. Basically JVM is divided in 3 parts as follows

Class Loader SubSystem

Various Memory Areas of JVM
Execution Engine



1. **Class Loader SubSystem**
   Java's dynamic class loading functionality is handled by the class loader subsystem. It loads, links and initialises the class when it refers to a class for the first time. Class loader subsystem is responsible for following 3 activities
   Loading : The Class loader reads the .class file, generates the corresponding binary data and save it in the method area.
   Linking : Performs verification, preparation, and (optionally) resolution.
   Initialization : In this phase, all static variables are assigned with their values defined in the code and static block(if any). This is executed from top to bottom in a class and from parent to child in class hierarchy.

2. **Memory Management Area**
   Java runtime Memory Area is divided into 5 parts.
   Method area : In 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.
   Heap area : Information of all objects is stored in the heap area. There is also one Heap Area per JVM. It is also a shared resource. Since the Method and Heap areas share memory for multiple threads, the data stored is not thread safe.

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 store methods calls. All local variables of that method are stored in their corresponding frame.The stack area is thread safe since it is not a shared resource. After a thread terminates, its run-time stack will be destroyed by JVM.
PC Registers : Each thread will have separate PC Registers, to hold the address of current executing instruction once the instruction is executed the PC register will be updated with the next instruction.
Native method stacks : For every thread, a separate native stack is created. It stores native method information.

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

Types of Variables

Based on the behaviour and position of declaration all variables are divided into the following 3 types.
1. Instance variables
2. Static variables
3. Local variables

Instance variable
- Instance variables will be created at the time of object creation and destroyed at the time of object destruction hence the scope of instance variables is exactly the same as scope of objects.
- Instance variables will be stored on the heap as the part of the object.
- Instance variables should be declared within the class directly but outside of any method or block or constructor.

Local variables
- Local variables will be stored inside the stack.
- The local variables will be created as part of the block execution in which it is declared and destroyed once that block execution completes. Hence the scope of the local variables is exactly the same as the scope of the block in which we declared.

var-args in java

- varargs (variable arguments) allow a method to accept a variable number of arguments of the same type.
- This feature was introduced in Java 5 to simplify the process of handling methods that need to accept a varying number of parameters.

new vs newInstance(), instanceof vs isInstance()

- **new vs newInstance():**new is a keyword used to create a new instance of a class. It is followed by a constructor invocation and allocates memory for the object on the heap. It is typically used when you know the specific class at compile-time.
- **newInstance()** is a method defined in the java.lang.reflect.Constructor class. It allows you to create an instance of a class using its default constructor or a constructor with specific arguments at runtime, even if you don't know the class name beforehand. It is primarily used for reflective operations.
- **instanceof vs isInstance():**
  instanceof is a Java operator used to test if an object is an instance of a particular class, a subclass of that class, or implements a particular interface. It evaluates to true if the object is of the specified type or a subtype, and false otherwise.