CORE JAVA

Day-3

CREATING JAVA PROGRAM IN NOTEPAD

- Open Notepad and write your java code
- Save the file with same name as that of class name.
- Open the command prompt and compile the program using javac command

```
javac demo.java
```

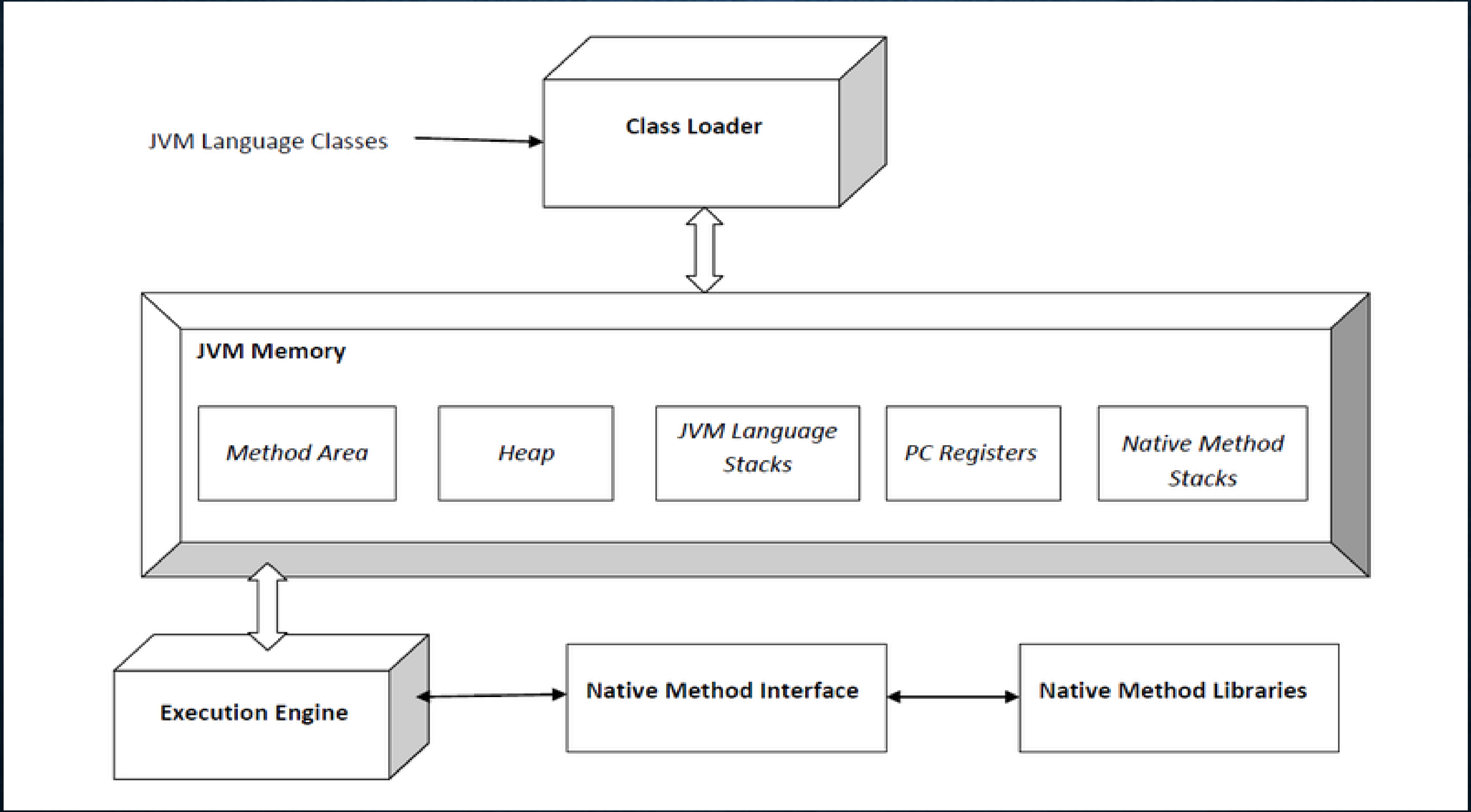
- Run the java program using java command

```
java demo
```

COMMAND LINE ARGUMENTS

- Command line arguments are values passed to a program when it is executed from the command line or terminal.
- The main purpose of command line arguments is to provide flexibility and allow users to control how a program behaves without having to recompile it or modify its source code.
- Some common use cases for command line arguments include:
  - Configuration
  - Input
  - Output
  - Control Flow
  - Automation

JVM ARCHITECTURE



CLASS LOADER

- The JVM (Java Virtual Machine) class loader is a crucial component of the Java runtime environment responsible for loading Java classes into memory as they are referenced by a Java program. The class loader performs the following main functions:
  - Loading: The class loader loads bytecode (compiled Java classes) into memory from various sources such as files, network, or other locations.
  - Linking: After loading the bytecode, the class loader links the classes by verifying the bytecode, preparing it for execution (by allocating memory for class variables and initializing the memory to default values), and resolving symbolic references to other classes.
  - Initialization: Finally, the class loader initializes the class by executing static initializers and static variable initializations.

METHOD AREA/METASPACE

- JVM memory model that stores class structures, method information, and metadata. It's a shared resource among all the threads running within the JVM and is used to store data that doesn't change during the execution of the program.
- JVM method area stores:
  - Class Metadata: Information about classes, interfaces, methods, and fields, including method signatures, access modifiers, and structural details.
  - Static Fields: Shared among all instances of a class, stored in the method area.
  - Constant Pool: Contains symbolic references to classes, methods, and fields, as well as literal constants.
  - Bytecode: Stores the bytecode of methods. When a class is loaded, its bytecode is parsed and kept in the method area.

HEAP AND STACK

- The "heap" refers to a specific area of memory where objects created by a Java program are stored. It's essentially a portion of memory dedicated to dynamically allocated memory for Java objects during runtime.
- The JVM language stack, often simply referred to as the "stack", is another important area of memory used for executing Java programs. It is utilized for
    - Method invocation
    - Stack frame
    - LIFO structure
    - Local variable and operation results

PC REGISTERS AND NATIVE METHOD STACK

- Program Counter (PC) Register:
  - The Program Counter (PC) register is a special register within the JVM that holds the address of the currently executing instruction. It keeps track of the execution point within the bytecode of the currently executing method.
- Native Method Stack:
  - The Native Method Stack is a memory area in the JVM used for executing native methods, which are methods written in languages other than Java (e.g., C, C++).

EXECUTION ENGINE

- The JVM execution engine is a crucial component responsible for executing Java bytecode instructions. It interprets the bytecode instructions and translates them into native machine code for the underlying hardware when necessary.
- In JDK 8 and later versions of the Java Virtual Machine (JVM), both interpretation and Just-In-Time (JIT) compilation are used, employing what's known as a mixed-mode execution strategy.
    - Interpreter: The interpreter is used initially to execute bytecode. When a Java application starts running, the JVM uses the interpreter to execute the bytecode instructions directly. This allows for quick startup and execution without waiting for compilation.
    - Just-In-Time (JIT) Compilation: As the application continues to run, the JVM identifies portions of code (hot spots) that are frequently executed. These hot spots are then compiled by the JIT compiler into optimized native machine code. This native code is then executed instead of interpreting the bytecode, resulting in improved performance for the frequently executed portions of the code.
- The JVM manages memory dynamically by allocating memory for new objects and reclaiming memory from objects that are no longer reachable or referenced by the application.

NATIVE METHOD INTERFACE AND LIBRARIES

- The Native Method Interface (JNI) and Native Method Libraries are mechanisms in Java that allow Java code to interact with code written in other programming languages, typically in C or C++.
- Native Method Interface (JNI):
  - The JNI is a programming framework that enables Java code running in the JVM to call and be called by native applications and libraries written in other languages, such as C or C++.
- Native Method Libraries:
  - Native method libraries are compiled libraries written in languages like C or C++ that contain implementations of native methods called from Java code.

CONTAINMENT

- Containment, in the context of object-oriented programming, refers to the concept of one class containing objects of another class as member variables.
- It's a way of structuring classes to represent relationships between them, where one class has a "has-a" relationship with another class.
- Eg. Car has an Engine, Courier has an address.