**JVM Architecture**

The **Java Virtual Machine (JVM)** is the engine that enables Java's **platform independence**, allowing Java programs to run on any device or operating system that has a JVM implementation.

Here's a breakdown of the **JVM architecture**:

**JVM Architecture Overview**

+-------------------------------------------+

| Class Loader Subsystem |

+-------------------------------------------+

↓

+-------------------------------------------+

| Runtime Data Areas |

| (Method Area, Heap, Stack, PC Register, |

| Native Method Stack) |

+-------------------------------------------+

↓

+-------------------------------------------+

| Execution Engine |

| (Interpreter, JIT Compiler, GC, etc.) |

+-------------------------------------------+

↓

+-------------------------------------------+

| Native Interface (JNI) |

+-------------------------------------------+

↓

+-------------------------------------------+

| Native Method Libraries |

+-------------------------------------------+

**Key Components Explained**

**1. Class Loader Subsystem**

* **Loads .class files** (bytecode) into memory.
* Performs:
  + **Loading**
  + **Linking** (verifying, preparing, resolving)
  + **Initialization**
* Ensures classes are loaded only once.

**2. Runtime Data Areas**

These are memory areas used during program execution:

**a. Method Area**

* Stores class-level data like:
  + Class structure
  + Field & method data
  + Static variables

**b. Heap**

* Stores **objects** and **instance variables**.
* Shared among all threads.

**c. Java Stack (per thread)**

* Stores:
  + Method frames
  + Local variables
  + Operand stacks
* Each thread has its own stack.

**d. Program Counter (PC) Register**

* Holds the **address of the current instruction** being executed per thread.

**e. Native Method Stack**

* Supports native methods written in **C/C++** (outside Java).
* Uses **JNI (Java Native Interface)**.

**3. Execution Engine**

Executes the bytecode.

**a. Interpreter**

* Reads and executes bytecode **line by line**.
* Slower but quick to start.

**b. JIT Compiler (Just-In-Time)**

* Converts bytecode to **native machine code**.
* Improves performance by caching frequently-used code.

**c. Garbage Collector**

* Automatically manages memory.
* Reclaims memory from unused objects in the heap.

**4. Native Interface (JNI)**

* Enables Java code to **interact with native applications** and libraries (written in C, C++, etc.).

**5. Native Method Libraries**

* System-level libraries (e.g., libc.so on Linux, msvcrt.dll on Windows).

**Summary**

| **Component** | **Role** |
| --- | --- |
| Class Loader | Loads .class files into JVM |
| Runtime Data Areas | Stores data, code, and objects |
| Execution Engine | Executes the program |
| JNI & Native Libs | Bridge between Java and native code |

------------------