**Execution Engine in JVM**

The **Execution Engine** is the **core component of the JVM** responsible for executing Java bytecode. It takes the bytecode (from .class files) and **converts it into machine code** that the CPU can understand.

**Components of the Execution Engine**

The **Execution Engine** consists of the following key components:

**1. Interpreter**

**What it does?**

* **Reads and executes bytecode line-by-line**.
* **Slower** because it **translates bytecode every time it's executed**.
* Used in the **initial phase** of execution.

**Example**

for (int i = 0; i < 1000; i++) {

System.out.println(i);

}

* The **interpreter executes each loop iteration separately**, translating the same bytecode multiple times, which makes it **slow**.

**2. HotSpot Profiler**

**What it does?**

* **Monitors** program execution to find **frequently used code** (hot spots).
* Determines which methods should be **optimized** by the JIT compiler.

**Example**

* If a method is executed **thousands of times**, HotSpot marks it as **"hot"**, and the JIT compiler **compiles it into native code**.

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

**What it does?**

* **Converts hot methods into native machine code** for faster execution.
* **Avoids re-interpreting bytecode** for frequently used methods.
* **Uses optimizations** like inlining, loop unrolling, and dead code elimination.

**How it Works (Step-by-Step)**

1. Initially, JVM **interprets all code**.
2. **HotSpot Profiler detects hot methods**.
3. **JIT Compiler compiles** these methods into **native machine code**.
4. The **JVM reuses the compiled code**, making execution faster.

**4. Garbage Collector (GC)**

**What it does?**

* **Automatically removes unused objects** to free memory.
* Runs in the **background** to optimize memory usage.
* Prevents **memory leaks**.

**Example**

String name = new String("Java");

// If `name` is no longer used, GC removes it from memory.

**How the Execution Engine Works (Complete Flow)**

**Step-by-Step Execution Process**

1️ **Class Loader loads .class files** into JVM.  
2️ **Execution Engine starts interpreting** the bytecode line-by-line.  
3️ **HotSpot Profiler analyzes execution** and detects frequently used (hot) methods.  
4️ **JIT Compiler compiles hot methods** into native code for better performance.  
5️ **Garbage Collector cleans up** unused objects from memory.