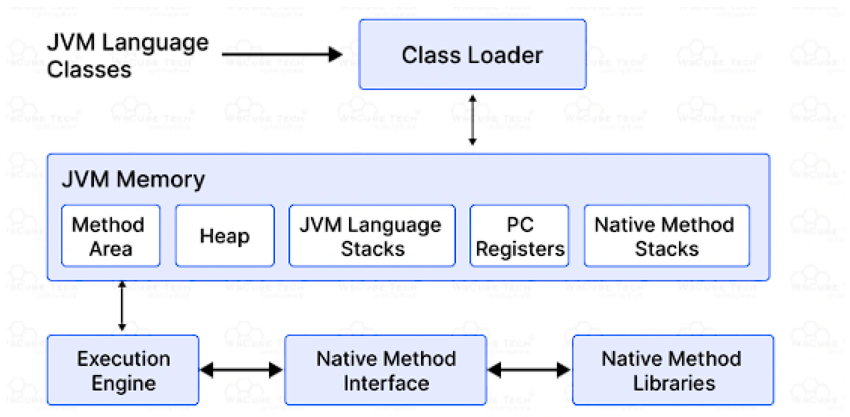
2 JVM vs JRE vs JDK (VERY IMPORTANT)

1. **What is JVM?**
    - JVM (Java Virtual Machine) is a part of Java that executes Java bytecode and allows Java programs to run on any device or operating system.
    - Key Responsibilities of JVM:
        - Loads class files
        - Verifies bytecode
        - Executes bytecode
        - Manages memory (Heap & Stack)
        - Performs Garbage Collection
    - JVM Working:
        - Java code is compiled into bytecode
        - JVM loads, verifies, and executes the bytecode
        - Execution is done using Interpreter + JIT Compiler
        - JVM manages memory and garbage collection



2. **What is JRE?**
    - JRE (Java Runtime Environment) provides the environment required to run Java applications.
    - Key Points:
        - JRE = JVM + Core Java libraries
        - It is used only to run Java programs
        - It cannot compile Java code
    - **NOTE: JRE is a runtime environment that includes JVM and libraries needed to execute Java programs.**

3. **What is JDK?**
    - JDK (Java Development Kit) is a software kit used to develop, compile, and run Java applications, which includes the JRE, compiler, JVM, and development tools.
4. **Difference between JVM, JRE, and JDK.**

Component	**Full Form**	**Purpose**	**Contains**
JVM	Java Virtual Machine	Runs Java bytecode on any platform	Only the runtime engine (executes .class files)
JRE	Java Runtime Environment	Runs Java applications	JVM + core libraries + supporting files (no compiler)
JDK	Java Development Kit	Develops, compiles, and runs Java programs	JRE + development tools (compiler, debugger, etc.)

- **NOTE:**
    - JVM: Runs Java programs.
    - JRE: JVM + libraries to run programs.
    - JDK: JRE + tools to write and compile programs.

5. **Can we run Java program without JDK?**
    - No, we cannot develop or compile a Java program without JDK, because JDK contains the compiler (javac).
    - However, if a program is already compiled (.class file), we can **run it using JRE alone**, since JRE has the **JVM**.
    - NOTE:
        - JDK → needed to write & compile.
        - JRE → enough to run pre-compiled Java programs.

6. **Can we run Java program without JVM?**
    - No, we cannot run a Java program without JVM, because JVM is responsible for executing Java bytecode on any platform.

7. **What is the role of JVM?**
    - The role of JVM (Java Virtual Machine) is to execute Java bytecode, making Java programs platform-independent.

- Key Points:
    - Loads .class files.
    - Verifies bytecode for security.
    - Executes code.
    - Manages memory (heap and stack).
- **NOTE: JVM runs Java bytecode, ensures platform independence, and manages memory during execution.**

8. Is JVM platform dependent or independent?

- JVM is platform-dependent because a different JVM is required for each operating system.
- Explanation:
    - Java code is platform-independent because it is compiled into bytecode.
    - JVM translates bytecode into machine code for the specific OS.
- **NOTE: JVM is platform-dependent, but Java bytecode is platform-independent.**

9. What are the components of JVM?

- The main components of JVM (Java Virtual Machine) are:
    i. **Class Loader:** Loads .class files into memory.
    ii. **Runtime Data Areas:** Memory areas used during execution, including:
        - Method Area – stores class structures.
        - Heap – stores objects.
        - Stack – stores method calls and local variables.
        - Program Counter (PC) Register – keeps track of instruction execution.
        - Native Method Stack – for native methods.
    iii. **Execution Engine:** Executes the bytecode.
    iv. **Native Interface (JNI):** Allows interaction with native applications (like C/C++).
    v. **Native Method Libraries:** Libraries needed for native method execution.
- **NOTE: JVM consists of Class Loader, Runtime Data Areas, Execution Engine, Native Interface, and Native Method Libraries.**

10. What is classloader?

- A ClassLoader in Java is a part of JVM that loads Java classes into memory at runtime.
- Key Points:

        - Loads .class files when required.
        - Follows a hierarchical delegation model:
            - Bootstrap ClassLoader – loads core Java classes (java.*).
            - Extension ClassLoader – loads classes from JDK extensions.
            - Application ClassLoader – loads classes from the application classpath.
- **NOTE: ClassLoader loads Java classes into memory dynamically at runtime.**