Java Architecture: JDK, JRE, and JVM Explained

Java Program Flow

.java files are written in High-Level Language.

These .java files are compiled using the Java Compiler (javac), producing a .class file.

The .class file contains Bytecode, which is executed by the JVM to convert it into machine code.

Flow Summary:

.java -> compile (javac) -> .class -> JVM -> Binary -> Run

JDK - Java Development Kit

Target Audience: Developers

Includes:

- JDK = JRE + Development Tools
  - Java Compiler (javac)
  - Java Debugger (jdb)
  - Javadoc, jar tools, etc.

JRE - Java Runtime Environment

Target Audience: End Users / Clients

Includes:

- JRE = JVM + Runtime Libraries (e.g., java.util, java.io)

JVM - Java Virtual Machine

Purpose:

- Executes the Bytecode (.class file).
- Provides Platform Independence.

Components of JVM

1. Class Loader Subsystem:

   - Class Loading: Loads .class files into JVM.
   - Verification: Ensures bytecode integrity and security.

2. Memory Management in JVM:
   - Method Area: Stores static variables, methods, class metadata.
   - Heap Area: Stores objects and instance variables.
   - Stack Area: Stores local variables, method call data (per thread).
   - Program Counter Register: Holds current and next instruction addresses.
   - Native Method Stack: Executes native methods (e.g., C/C++ via JNI).

3. Execution Engine:
   - Interpreter: Executes bytecode line by line.
   - JIT Compiler: Converts frequently used bytecode into native code for speed.
   - Garbage Collector: Frees memory by removing unused objects.

Summary Table

JDK = JRE + Development Tools

JRE = JVM + Libraries

JVM = Bytecode Runner and Manager

Memory Components:

- Method Area: Class metadata and static members
- Heap: Objects and instance data
- Stack: Method call stack per thread
- PC Register: Instruction pointer
- Native Stack: Native method execution

Execution Components:

- Interpreter: Line-by-line execution
- JIT Compiler: On-the-fly optimization

Java Architecture: JDK, JRE, and JVM Explained

- Garbage Collector: Memory management