Java Essentials: From What Programming Is to How Java Works

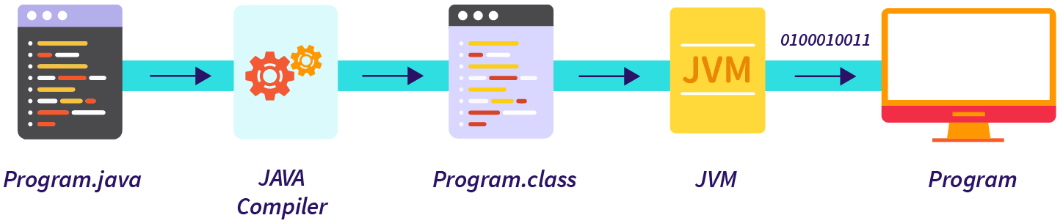
What Programming Is ?

Programming is the process of giving step-by-step instructions to a computer to perform specific tasks. Before diving into Java, it's important to understand what a **program** (a set of instructions), **code** (the written commands), and a **compiler** (a tool that translates code into machine language) are. Also, get familiar with **syntax** — the set of rules that define the correct structure of code, much like grammar in human languages.

Key Features of Java :

1. **Simple & Readable** – Clean syntax, easy to learn, avoids complex concepts like pointers.
2. **Object-Oriented** – Everything is treated as an object, promoting modular and reusable code.
3. **Platform-Independent** – Code runs on any system via the JVM ("write once, run anywhere").
4. **Secure** – Runs in a controlled JVM environment, protecting against unauthorized access.
5. **Robust** – Strong memory management, built-in garbage collection, and exception handling.
6. **Multithreaded** – Supports concurrent execution for better resource utilization.
7. **High Performance** – JIT compiler optimizes bytecode into fast native machine code.
8. **Distributed** – Built-in support for networking and remote method invocation (RMI).

How Java Works



---

1. Java Code is Written (.java file)

You write Java code using a text editor or IDE. This source code is saved with a .java extension.

2. Compilation into Bytecode
The Java Compiler (javac) takes your .java file and compiles it into **bytecode**, which is saved as a .class file.

👉 Bytecode is an intermediate code that is not human-readable but can be understood by the **Java Virtual Machine (JVM)**.

3. Bytecode is Platform-Independent
This bytecode can run on **any system (Windows, Linux, macOS, etc.)**, as long as a JVM is installed.
This is why Java is known as "**Write Once, Run Anywhere**".

4. JVM is Platform-Dependent
While the bytecode is platform-independent, the **JVM itself is platform-specific** (different versions for different operating systems).
Each platform has its own version of the JVM to interpret and run the same bytecode.

5. JVM Loads and Executes Bytecode
The JVM does the actual work of:

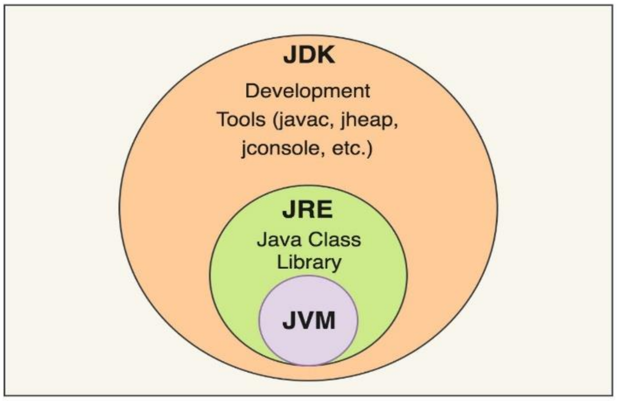
- **Class Loading** (via Class Loader)
- **Bytecode Verification** (to ensure no harmful code runs)
- **Code Execution** using the **Interpreter** or **JIT Compiler**

6. Just-In-Time (JIT) Compilation
Frequently used (hotspot) bytecode is converted to **native machine code** by the **JIT Compiler** for faster execution.

7. Runtime Environment
The Java Runtime Environment (JRE), which includes the JVM and core libraries, provides all the necessary resources for running Java applications.

Java Environment: JDK, JRE & JVM

To run and write Java programs, you need these three core components:



✅ JDK (Java Development Kit)

- Complete package for **developing Java applications**
- Contains: JRE + Development tools (like compiler javac, debugger, etc.)

✅ JRE (Java Runtime Environment)

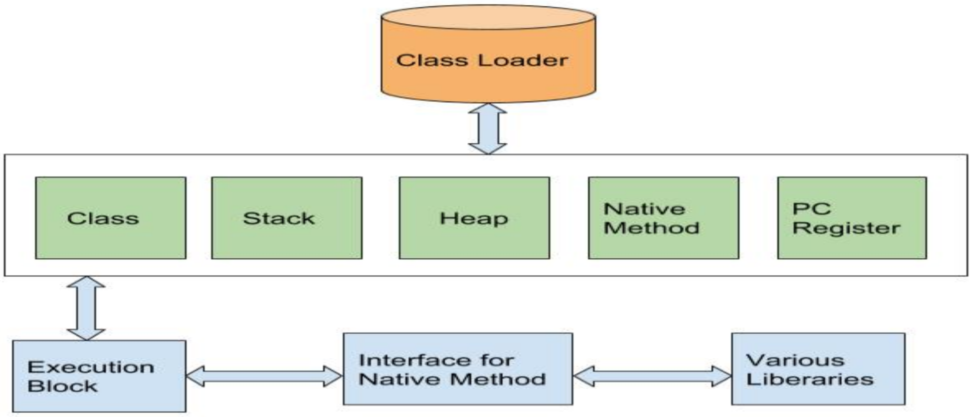
- Used to **run** already compiled Java programs
- Contains: JVM + necessary libraries and classes

✅ JVM (Java Virtual Machine)

- Core engine that **executes Java bytecode**
- Makes Java **platform-independent**

JVM (Java Virtual Machine) Architecture

📝 Java follows the principle: **Write Once, Run Anywhere** thanks to the JVM.



1. Class Loader Subsystem

Loads .class files into the JVM in three steps: **Loading**, **Linking**, and **Initialization**. Ensures classes are loaded only once and when needed

2. Runtime Data Areas (Memory)

- **Method Area:** Stores class-level info like methods, static variables, and constants.
- **Heap:** Shared memory space for all objects and instance variables.
- **Java Stack:** Holds method calls, local variables; each thread gets its own stack.
- **PC Register:** Keeps track of the current instruction per thread.
- **Native Method Stack:** Supports execution of native (non-Java) code.

3. Execution Engine

- **Interpreter:** Executes bytecode line by line.
- **JIT Compiler:** Boosts performance by compiling frequently used code to native machine code.
- **Garbage Collector:** Frees memory by removing unused objects automatically.

4. JNI (Java Native Interface)

Enables Java to call and use native code written in C/C++.

5. Native Libraries

External platform-specific libraries (.dll/.so) used via JNI.