JAVA FUNDAMENTALS – DETAILED EXPLANATION

---

1. What is a Platform

A **platform** is the **environment in which a program runs**.

A platform is made up of:

- **Hardware** such as processor, memory, motherboard
- **Software** such as operating system (Windows, Linux, macOS)

Examples:

- **Intel processor + Windows OS = one platform**
- **Intel processor + Linux OS = another platform**
- **Apple Silicon + macOS = a different platform**

Important point:
Even if the hardware is the same, changing the operating system creates a different platform because the OS controls how software interacts with hardware.

---

2. Platform Dependent Language

A **platform dependent language** produces programs that run only on a specific operating system.

Example languages:

- **C**
- **C++**

Why C and C++ are Platform Dependent

When a C or C++ program is compiled:

- The compiler converts source code directly into **machine-level instructions**
- These instructions are specific to:
    - The operating system
    - The processor architecture

Examples:

- Windows generates **.exe** files

- Linux generates **ELF** executables
- macOS generates **Mach-O** executables

A Windows executable cannot run on Linux or macOS.
Therefore, the same program must be compiled separately for each platform.

---

3. Why Companies Share Only Executable Files

Companies do not share source code because:

- Source code exposes business logic
- Anyone can copy or modify it
- Security risks increase

Instead, companies:

- Compile the source code
- Share only executable files

That is why the same application has different installers for Windows, Linux, and macOS.

---

4. Platform Independent Language

A **platform independent language** allows a program to run on multiple operating systems without changing the source code.

Java is a platform independent language.

However, Java does not run directly on the operating system.
It uses an additional layer to achieve portability.

---

5. Core Idea Behind Java

Java introduced a middle layer between:

- Source code
- Machine-level code

This middle layer is called **bytecode**.

Instead of generating OS-specific machine code, Java generates **OS-neutral bytecode**, which can run on any system that has a JVM.

---

6. Java Architecture

Java consists of the following components:

JDK (Java Development Kit)

Used for development.
Contains:

- Java compiler (javac)
- Development tools
- JRE

JRE (Java Runtime Environment)

Used to run Java programs.
Contains:

- JVM
- Core Java libraries

JVM (Java Virtual Machine)

- Executes Java bytecode
- Converts bytecode into machine-level instructions
- Is platform dependent

Important fact:

- **Bytecode is platform independent**
- **JVM is platform dependent**

---

7. Java Compilation and Execution

Step 1: Writing Source Code

You write Java code in a file such as:

```
Demo.java
```

This code is human-readable and high-level.

Step 2: Compilation

When you run:

```
javac Demo.java
```

The compiler:

- Checks syntax
- Converts source code into a `.class` file
- Generates **bytecode**, not machine code

Step 3: Execution

When you run:

```
java Demo
```

The JVM:

- Reads bytecode
- Converts it into machine instructions
- Sends them to the processor for execution

This is why Java is called:

- **Compiled** (because of javac)
- **Interpreted** (because JVM executes bytecode)

---

8. How Java Achieves Platform Independence

- Java code is written once
- It is compiled once into bytecode
- The same bytecode is shared across systems

Each operating system has its own JVM:

- Windows JVM
- Linux JVM
- macOS JVM

Each JVM converts the same bytecode into its own machine-level code.

This concept is called **Write Once, Run Anywhere (WORA)**.

---

9. Role of JVM

The JVM acts as:

- A virtual machine
- A bridge between bytecode and hardware

Without JVM:

- Java would behave like C/C++
- Separate executables would be needed for each OS

---

10. Java Execution Flow

Demo.java
→ javac
→ Demo.class (bytecode)
→ JVM
→ Processor
→ Output

---

11. Java History

- Java started in **1991**
- Developed by the **Green Team**
- Led by **James Gosling**
- Created at **Sun Microsystems**

Initial goal:

- Build a language that could run on multiple devices and platforms

---

12. Java Version Evolution

- Java 1.0 released in **1995**
- Versions 1.2 to 1.5 followed old naming style
- From Java 6 onwards, simple numbering was used
- Current stable version is **Java 21**

---

13. Why Java Became Popular in Enterprises

Java succeeded because of **standardization**.

Java defined:

- Interfaces and abstract rules

Database vendors implemented:

- Their own logic based on Java standards

This led to:

- JDBC
- Portable database applications
- One Java program working with multiple databases

---

14. Oracle and Java

- Oracle acquired Sun Microsystems
- Ownership of Java moved to Oracle
- Licensing policies changed
- Java remained widely used and free for development

---

15. Final Takeaway

- A platform is hardware plus operating system
- C and C++ are platform dependent
- Java uses bytecode instead of direct machine code
- JVM makes Java platform independent
- Bytecode is universal
- JVM is OS-specific
- Java follows Write Once, Run Anywhere
