# Day 1

## Introduction To Java

### ✨ Introduction to Java (In Simple Language)

Java is a *high-level, object-oriented programming language*. It was first developed by Sun Microsystems in 1991, and later acquired by Oracle Corporation.

### 💡 What does "High-Level" Mean?

Java is called a high-level language because it is **easy to read and write for humans**. Unlike machine code or assembly language, Java uses words and symbols we can understand (like if, while, class).

### ⚙️ What does "Object-Oriented" Mean?

Java is object-oriented, which means **everything in Java is based on objects and classes**. It helps to organize code and make it reusable, cleaner, and easier to manage.

## History of Java's Name:

- **The Initial Name – Oak:**
  - In 1991, Java was originally developed by James Gosling and his team at Sun Microsystems.
  - The language was first called Oak, named after an oak tree that stood outside Gosling's office. The developers chose this name because they felt it symbolized strength and stability.
  - The original project aimed to create software for interactive television, which was an emerging field at the time.

- **The Name Change:**
  - When Sun Microsystems began preparing the language for release, they discovered that Oak was already the trademarked name of another programming language.
  - As a result, they needed to rename the language to avoid trademark issues.

- **The New Name – Java:**
  - In 1995, the language was rebranded as Java.

- The new name was inspired by Java coffee, a popular coffee from Indonesia. The developers were fond of drinking this coffee, and the name symbolized energy, vitality, and a fresh start.
- "Java" was chosen because it was short, catchy, and symbolized the excitement they wanted the language to evoke.

## 🚀 Why is Java So Popular?

- **Platform Independent**: Java follows the “Write Once, Run Anywhere” principle. This means you can write Java code once, and it will run on any computer that has Java Virtual Machine (JVM) installed.
- **Secure and Reliable**: Java has built-in features to protect your code and system. It is also known for its stability and reliability, which is why banks, enterprises, and governments still use it.
- **Versatile (Can be used everywhere):**
  - Mobile apps (especially Android apps)
  - Web applications (like websites and web servers)
  - Desktop software (like calculators or media players)
  - Games

- Enterprise systems (used in big businesses)

# Features of Java:

**Simple**: Easy to learn if you understand basic programming.

**Object-Oriented:** Everything revolves around objects and classes.

**Platform Independent:** Thanks to the JVM, Java code can run anywhere.

**Robust**: Robust means strong, reliable, and able to handle errors without crashing easily. Java is called a robust language for several reasons:

- **has strong memory management** i.e. has an automatic garbage collector that removes the unused objects from memory - reduces memory leaks and improves performance.
- **has a powerful system to catch and handle errors** and exceptions so that the program doesn't crash suddenly.
  (**Error** means something went wrong that your program **cannot fix**, while **Exception** means something went wrong that your program **can handle**.)

**Secure**: It includes safety features to prevent hacking or unauthorized access.

**Multithreaded**: Java can do many tasks at the same time (multitasking).

**Portable**: Java programs can move easily from one system to another.

**High Performance**: Though not as fast as C/C++, Java is faster than many other high-level languages because of its Just-In-Time (JIT) compiler.

- **JIT** is a tool inside the **JVM**.
- It **converts Java bytecode into machine code just before** our program runs.

## □ How Java Works (Basic Concept)

- We write Java code (.java file).
- The Java compiler(javac) turns it into bytecode (.class file).
- The Java Virtual Machine (JVM) reads the .class file.
- JVM converts the bytecode into machine language that your computer understands.
- Then, the computer runs the program and gives us the output.

This system makes Java portable, efficient, and flexible.

# Java Tokens:

## What are Java Tokens?

- Java Tokens are the smallest meaningful units in a Java program.
- Just like words make sentences, tokens make Java programs.
- Whenever we write any Java code, it is made up of tokens, and they are recognized by the Java compiler.

## Types of Java Tokens:

Java has 6 main types of tokens:

1. **Comments:**
    - Comments are notes written in code to explain what it does.
    - They are ignored by the compiler and are just for human understanding.

    **Single-line comment**: starts with //
    Example:
    // This line shows the student's marks

**Multi-line comment**: starts with /* and ends with */

Example:

```
/* This is a comment
   that spans multiple lines */
```

2. **Identifiers:**
    - Identifiers are names used for things like variables, classes, and methods.
    - You choose these names, but they must follow some rules.

**Rules**:

- Can’t be a Java keyword (like class, int)
- Must start with a letter, $, or _
- Can have numbers after the first character
- Case-sensitive (Marks and marks are different)

Example:

```
int marks;  // "marks" is an identifier
String name; // "name" is also an identifier
```

3. **Keywords:**

- Keywords are predefined words in Java with special meanings.
- We cannot use them as names for variables or classes.

Examples of keywords:
int, class, public, static, void, if, else, for, while, etc.

Example in code:

```
public class HelloWorld {
    public static void main(String[] args) {
        int age = 20;
        if (age > 18) {
            System.out.println("Adult");
        }
    }
}
```

4. **Literals:**
   Literals are fixed values used directly in code.

   Example of different literals:
   Numbers: 100, 3.14
   Characters: 'A', 'z'
   Strings: "Hello", "Java"
   Boolean: true, false

Example in code:

```
int age = 18;          // 18 is a literal
String name = "Ram";   // "Ram" is a literal
```

5. **Operators:**
   Operators are symbols used to perform operations like math, comparison, and logic.

   Examples:
   Arithmetic: +, -, *, /, %
   Comparison: ==, !=, <, >, <=, >=
   Logical: &&, ||, !

   Example in code:

```
int a = 10, b = 20;
int sum = a + b; // "+" is an arithmetic operator
```

6. **Separators:**
   Separators are symbols used to structure the code and separate parts of the program.

   Examples:
   () – for method calls or conditions
   {} – to group code blocks
   [] – for arrays

; – ends a statement
, – separates multiple variables
. – accesses object members (dot operator)

Example in code:

```
public static void main(String[] args) {
    int[] numbers = {1, 2, 3};  // {} and [] are separators
    System.out.println(numbers[0]); // . is used to call a method
}
```

# Java Editor Softwares:

To write and run Java programs easily, we use special software called an **IDE** (Integrated Development Environment). It helps us write, check, and run Java code all in one place.

Two popular IDEs for Java are:

**NetBeans**

- Free and open-source
- Good for beginners
- GUI design made easy
- Smart code suggestions — It helps complete our code and shows hints to fix errors.

- Built-in debugger — Helps find and fix mistakes in our program.

**Eclipse**

- Used a lot in big companies — Professionals often use Eclipse for large projects.
- We can add plugins (extra tools) to do more tasks.
- Great for big Java projects — Handles large amounts of code well.
- Supports many languages not just Java—can also work with C, C++, Python, etc.

# Java Editions:

1. **JSE (Java Standard Edition):**
   **Used For:**
   - Creating basic Java programs
   - Building desktop applications
   - Learning core Java concepts

   What it Includes:
   - Core libraries like java.lang, java.util, java.io, etc.

- Tools for object-oriented programming, multithreading, file handling, exception handling, etc.

Example:
A simple calculator app or a desktop notepad made using Java.

2. **JEE (Java Enterprise Edition):**

**Used For:**

- Developing large-scale, distributed, and web-based applications
- Common in banking, e-commerce, enterprise apps

**What it Includes (in addition to JSE):**

- Servlets – for handling requests on a web server
- JSP (Java Server Pages) – for dynamic web content
- EJB (Enterprise JavaBeans) – for business logic in big apps
- JPA (Java Persistence API) – for managing data in databases

Example:
A bank's online portal where users can log in, view account balance, and make transactions.

3. **JME (Java Micro Edition):**

**Used For:**

- Developing apps for small devices
- Embedded systems, older mobile phones, smart cards, TVs, etc.

What it Includes:

- Lightweight versions of Java libraries
- Special APIs for limited memory and less powerful hardware

Example:

Games or apps on old Nokia phones or software in digital meters.

4. **JFX (JavaFX):**

**Used For:**

- Creating modern, rich graphical user interfaces (GUIs)
- Apps with animations, media players, and visual effects

What it Includes:

- Tools to design GUI layouts, handle mouse events, animations, charts, and audio/video

Example:

A music player app with attractive design and animation made using JavaFX.

# Components of Java Platform:

1. **JDK – Java Development Kit**

   **What it is:**

   A complete toolkit for Java developers.

   **Purpose**:

   Used to write, compile, and run Java programs.

   **Contains**:

   - JRE (Java Runtime Environment)
   - Java compiler (javac)
   - Development tools (debugger, documentation tools, etc.)

   **Use case:**

   If we want to *create Java programs*, we need the JDK.

2. **JRE – Java Runtime Environment**

   **What it is:**

   A package used to run Java applications.

   **Purpose:**

   Allows our computer to *run Java programs*, but **not** to write or compile them.

   **Contains:**

- JVM (Java Virtual Machine)
- Class libraries (pre-built Java classes)

**Use case:**

If we just want to use a Java-based application (not build one), you only need the JRE.

3. **JVM – Java Virtual Machine**

   **What it is:**

   A virtual machine that runs Java bytecode (compiled Java code).

   **Purpose**:

   - Converts Java bytecode into machine code (understandable by your CPU).
   - Makes Java programs platform-independent (runs on Windows, Linux, Mac, etc., without changes).

   **Use case:**

   Every time we run a Java program, the JVM is what actually executes it.

4. **API – Application Programming Interface**

   **What it is:**

   A set of pre-written Java classes and packages.

   **Purpose:**

   - Lets developers reuse existing code instead of writing everything from scratch.

- Makes development faster and easier.

**Examples:**

- java.util → for data structures like ArrayList, HashMap
- java.io → for input and output operations
- java.lang → for core language features (like Math, String, System)