Java OOP — Unit 1

What is Java?

- Java is a high-level, object-oriented programming language.
- Used in mobile apps (Android), web apps, desktop apps, games, and enterprise systems.
- **Key idea:** Write once, run anywhere (WORA).

Example (basic program):

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, Java!");
    }
}
```

Short History of Java

- Developed in 1995 by James Gosling at Sun Microsystems.
- Originally designed for smart appliances, but became popular for the web.
- Now widely used for Android and enterprise software.

Java Buzzwords (Main Features)

- **Simple:** Easy to learn, syntax like C/C++ but simpler.
- **Object-Oriented:** Based on classes and objects.
- **Portable & Platform-independent:** Code runs anywhere with JVM.
- **Robust & Secure:** Strong error checking and memory management.
- **Multithreaded:** Can run multiple tasks at once.
- **High Performance:** JIT compiler makes it faster.

Java Virtual Machine (JVM)

- Runs Java bytecode on any device.
- Converts bytecode into machine instructions.
- Makes Java platform-independent.

Real-world analogy: JVM = translator that makes sure your language (Java) is understood in any country (machine).

Java Runtime Environment (JRE)

- Includes JVM + standard Java libraries.
- Needed to **run** Java programs.

Bytecode

- Java compiler (`javac`) converts `.java` files into `.class` files (bytecode).
- JVM executes bytecode.

Analogy: Bytecode is like a universal recipe; JVM is the chef in any kitchen.

Object-Oriented Programming (OOP) Principles

- **Class:** A blueprint (like a car design).
- **Object:** A real-world instance (like an actual car).
- **Encapsulation:** Keeping data safe (use `private` + getters/setters).
- **Inheritance:** Child class inherits from parent (e.g., ElectricCar extends Car).
- **Polymorphism:** One action, many forms (different `drive()` methods).
- **Abstraction:** Hide details, show essentials (e.g., driver uses steering wheel, not engine mechanics).

Example (class & object):

```
class Dog {
    String name;

    void bark() {
        System.out.println(name + " barks loudly");
    }
}

public class TestDog {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.name = "Bolt";
        d.bark();
    }
}
```

Writing Simple Java Programs

- Write code in `FileName.java`.
- Compile: `javac FileName.java`
- Run: `java FileName`

Every program must have a * `main` *** method.**

```
public class SimpleExample {
    public static void main(String[] args) {
        System.out.println("First Java program running!");
    }
}
```

Compiling and Running

- **Command Line:**

```
javac Hello.java
java Hello
```

- **IDE (Eclipse, IntelliJ, VS Code):** Press run button.

Command Line Arguments

- Pass values when running program.

```
public class CmdArgsExample {
    public static void main(String[] args) {
        if (args.length > 0) {
            System.out.println("Hello, " + args[0]);
        } else {
            System.out.println("No arguments provided");
        }
    }
}
```

Run:

```
java CmdArgsExample Aliza
```

Output: `Hello, Aliza`

Using Scanner for Input

- Reads input from keyboard.

```
import java.util.Scanner;

public class InputExample {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter your name: ");
        String name = sc.nextLine();

        System.out.print("Enter your age: ");
        int age = sc.nextInt();

        System.out.println("Welcome, " + name + ". Age: " + age);
    }
}
```

System.out.print vs println

- `print()` → prints without moving to next line.
- `println()` → prints and moves to next line.

```
public class PrintExample {
    public static void main(String[] args) {
        System.out.print("Hello");
        System.out.print(" World");

        System.out.println("!");
        System.out.println("This is a new line.");
    }
}
```