BASIC INTRODUCTION OF OOPs AND JAVA

The most important and basic sentences you should memorize to understand object-oriented concepts in Java.

- **Object-Oriented Programming (OOP)** is a way to organize code using real-life objects.
- A **class** is like a blueprint, template or recipe for creating objects (e.g., a "Car" class).It defines what data (fields) and actions (methods) the objects will have.
- An **object** is an actual thing created from a class (e.g., a specific car). It represents something real we can work with in our code.
- **Fields** (or variables) are data, properties or information that objects store. For example, a field for a "Car" class could be "color" or "speed."
- **Methods** are a group of instructions (like a small program) inside a class that does something or performs an action, such as "drive" or "brake."
- **Encapsulation** means keeping data safe inside a class and hiding details—programmer controls how it's used through methods. Data is kept inside a class safe from outside changes by using "private" fields and allowing access only through valid methods.
- **Inheritance** lets one class use the features (fields and methods) of another class (e.g., "SportsCar" inherits from "Car").
- **Polymorphism** means objects can take many forms: one method name can work differently depending on the object.
- **Abstraction** means showing only what's important and hiding complex details. We don't have to understand the electronics to watch TV, this is abstraction.
- **Package:** A package is a group of related classes. It helps organize our code into folders.

Here are some basic yet essential things to know about a typical Java program.

- **Every Java program is made inside a class.** The class is like a container for our code, and its name should start with an uppercase letter.
- **The main method is where the program starts running.** It looks like this:

```
public static void main(String[] args) {
        // code goes here
}
```

Everything inside the main method's curly braces {} gets executed when we run our program.

- **A simple Java program structure usually includes these parts:**
  - Documentation/comments (optional, for notes in our code)
  - Package statement (optional, to organize programs into folders)
  - Import statements (optional, to use built-in Java classes)
  - Class declaration (the main part — always needed)
  - Main method (always needed to run the program)
  - Variables (to store data)
  - Methods (for actions your program does)

- **Java is case-sensitive** — for example, 'Main' and 'main' are different.

- **File name and class name should match.** If your class is named Main, the file should be named Main.java.

- **Example of the simplest Java program:**

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

Save: Main.java
Compile: javac Main.java
Run: java Main

The essential components in a basic Java program structure are:

- **Package Declaration** (optional): This organizes related classes into namespaces and must be the first statement if present. Example: *package mypackage;*.
- **Import Statements** (optional): Allow us to use classes from other Java libraries. Placed after the package declaration. Example*: import java.util.Scanner;*.
- **Class Declaration**: Every Java program must have at least one class. This is the main building block, and all code is contained inside the class body. Example*: public class MyProgram {*
- **Main Method**: The entry point of every Java application. The program starts running from here. Its signature must be: *public static void main(String[] args) {*
- **Variables**: Used to store data, declared inside classes or methods.
- **Statements and Expressions**: The actual instructions to be executed, written inside methods such as *main()*.
- **Methods**: Blocks of code that perform actions. Besides the main method, we can define other methods to organize the behavior of the class. Methods interact with each other to perform the task via exchange of data (arguments/parameters).
- **Documentation/Comments**: Used for explaining the code, making it easier to understand for humans.
- **Constructor**: A special method to initialize objects when they are created.

The main method is crucial in Java programs because it acts as the **entry point** where the Java Virtual Machine (JVM) starts the execution of our code. Without the main method, a standalone Java program will not run—if we omit it or define it incorrectly, the JVM cannot find where to begin and the program will not execute.

Key reasons why the main method is essential:

- **Program Entry Point:** The main method (with the signature *public static void main(String[] args)*) is the first place the JVM looks for instructions to execute in a standalone Java application.
- **Standardized Starting Place:** Its specific signature tells the JVM exactly where and how to start running the code.
- **Static for Direct Access:** Declaring *main* as *static* allows the JVM to call it without creating an object of the class, which is necessary since no objects exist at the start of execution.
- **Handles Inputs:** The *String[] args* parameter lets your program receive command-line input when it starts, adding flexibility of providing data before the program runs,.
- **Program Control:** All further method calls and logic execution in a Java console application flow from the main method, so it provides a centralized control point for starting and managing execution.

If the main method is missing or its signature is changed, the program may compile but will not execute, because the JVM cannot locate the required starting point.

The JVM recognizes the **main method as the program's entry point** because it looks specifically for a method with the exact signature:

public static void main(String[] args)

This is why and how the JVM identifies it as the starting point:

- **Method Name:** The JVM strictly searches for a method named `main`. This specific name is the identifier used by the JVM to start execution.
- **Public Access Modifier:** The method must be `public` so that the JVM, which is external to your class, can access and invoke this method.
- **Static Modifier:** It must be *static*, allowing the JVM to call it without creating an instance (object) of the class. This is important because when the program starts, no objects exist yet.
- **Void Return Type:** Its return type must be *void* because the JVM does not expect any return value from the starting point.
- **Parameter:** The parameter must be a single array of `String` *(String[] args)*, enabling the program to receive command-line arguments if any.

When we run our Java application, the JVM loads the class, looks for this specific method signature, and calls it to begin running our program. The *main* method then acts as the centralized starting point from which all other methods and statements are executed. The program is terminated when the last statement of *main()* is executed.