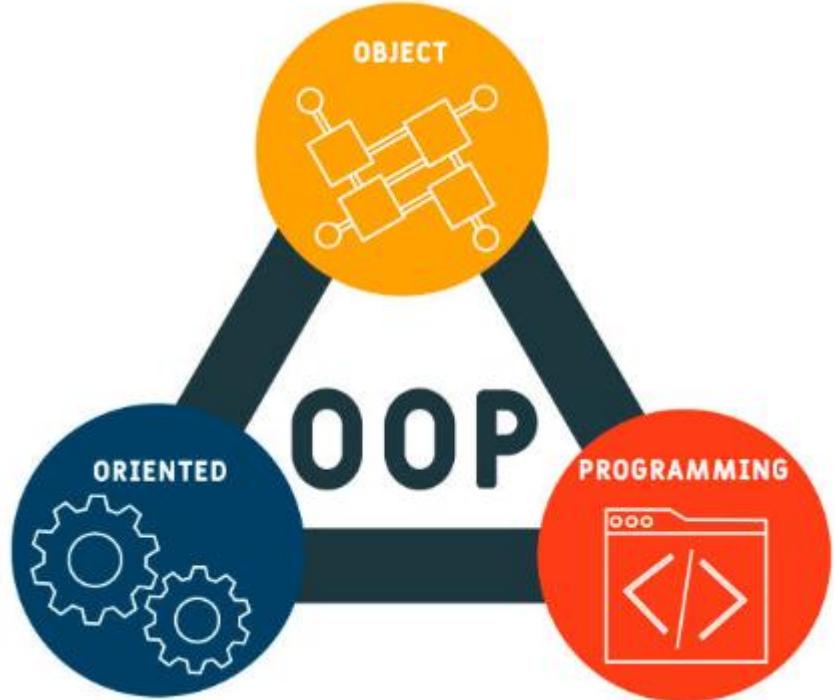


# OOP Concepts and Implementation



Nimesha Hewawasam  
Lecturer  
[nimesha.h@nsbm.ac.lk](mailto:nimesha.h@nsbm.ac.lk)

# Content

- Object Oriented Programming
- OOP Hierarchy
- Files, Folders and Packages
- Classes
- Class Scope
- Variables
- Objects
- Constructors
- Object Generation
- Methods

# Introduction to OOP in Java

- Java is an **Object-Oriented Programming Language**.
- OOP organizes code into **classes** and **objects**.
- Use **Multi Threading**
- Benefits:
  - Code reusability
  - Modular design
  - Easier maintenance

# OOP Hierarchy

- Classes
- Objects
- Methods
- Variable
- Those are only things we need to do program.

# Files, Folders and Packages

- **Class:**
  - source – *classname.java*
  - compiled – *classname.class*
- **Package:**
  - collection of classes, like a library
  - package name – all lower case
  - source package – in project folder, in subfolder `src\packagename`
  - compiled – in project folder, in subfolder `build\classes\packagename`

# First application with Netbeans

**Hello\_World\_App (Project Root):** This is your **main project folder**. It contains everything related to your Java application.

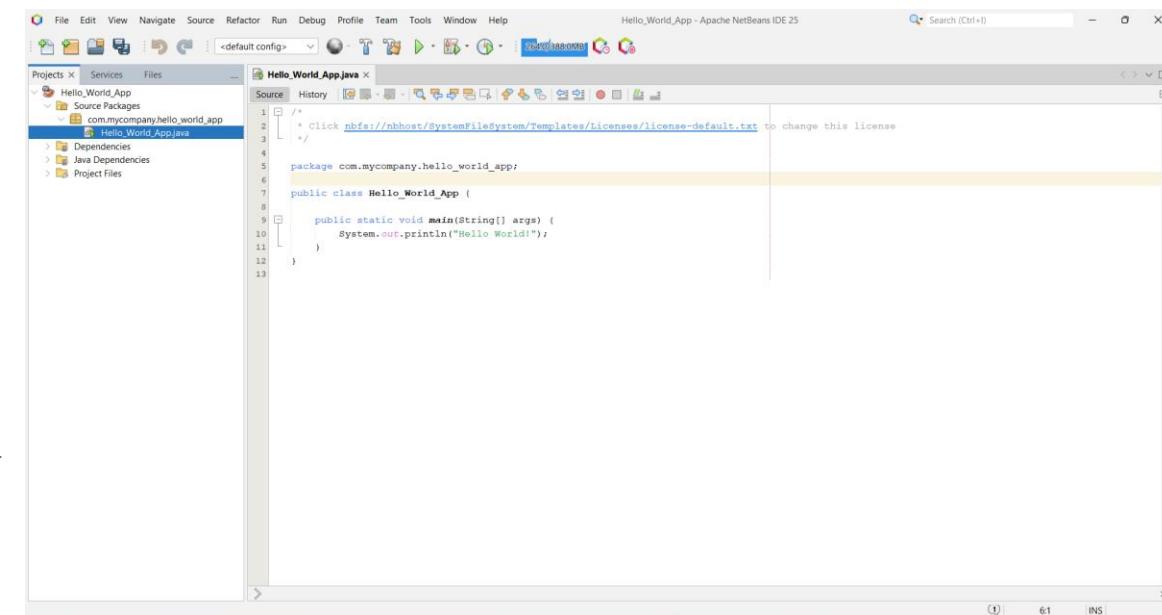
**Source Packages:** This folder holds your **Java source code files**, structured by **packages**.

**com.mycompany.hello\_world\_app:** is the package name

**Hello\_World\_App.java:** is your **Java class file** that contains the `main()` method

**Dependencies / Java Dependencies :** These folders show any external **libraries** or **Java platform dependencies** your project relies on. For a simple "Hello World", this remains default (i.e., standard Java libraries).

**Project Files:** This contains **metadata files** used by NetBeans to manage your project (like build configs, project settings, etc.). You usually don't need to edit these manually.



# First application with Netbeans

**Hello\_World\_App (Project Root)** : This is your entire Maven project folder.

**src/main/java** : Standard Maven project structure for source code.

**com/mycompany/hello\_world\_app/Hello\_World\_App.java** : Your source code file. This is the Hello World class you wrote.

**Target**: This is the output directory Maven uses to store compiled files and other generated content.

**Inside target**: classes/com/mycompany/hello\_world\_app/Hello\_World\_App.class. The compiled .class file for your Java source (.java) file.

**generated-sources/annotations/** : Reserved for annotation processors (e.g., if you're using frameworks like Lombok or JPA in advanced cases).

**maven-status & maven-compiler-plugin/compile/** : Internal Maven build system files for tracking compile actions and input/output.

**pom.xml** : This is your Maven configuration file.

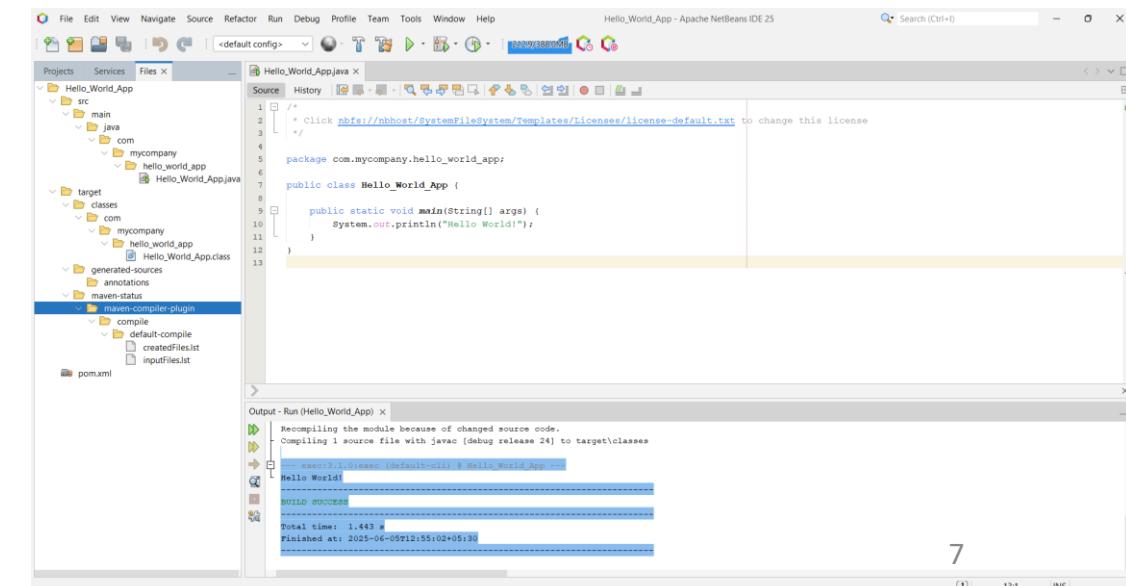
It declares project metadata and dependencies.

For now, it's very simple — but later, you'll use this to:

- Add libraries (e.g., JUnit, Spring Boot)

- Define Java version

- Configure plugins



# First application with Netbeans

The figure consists of three vertically stacked screenshots of the Apache NetBeans IDE 25 interface.

**Screenshot 1 (Top):** Shows the "New Project" wizard. The "Choose Project" step is selected. Under "Categories", "Java with Maven" is chosen. The "Projects" list includes "Java Application", "Web Application", "EJB Module", "Enterprise Application", "Enterprise Application Client", "OSG Bundles", "Micronaut Project", "NetBeans Module", "NetBeans Application", "Payara Micro Application", "FXML JavaFX Maven Archetype", and "Simple JavaFX Maven Archetype". A description below states: "A simple Java SE application using Maven. You are recommended to begin here, if you are new to Java!". The "Output - Run (NB\_1)" panel shows the build process: "Recompiling 1 source file with javac [debug release 24] to target\classes", "Compiling 1 source file with javac [debug release 24] to target\classes", and "exec:java: Default-Cls # Hello\_World\_App -> Hello\_World". The status bar at the bottom indicates "BUILD SUCCESS".

**Screenshot 2 (Middle):** Shows the completed project structure for "Hello\_World\_App". The "Projects" panel lists "Hello\_World\_App", "Source Packages", and "com.mycompany.hello\_world\_app". The "Files" panel shows the "Hello\_World\_App.java" file content:

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 */
package com.mycompany.hello_world_app;

public class Hello_World_App {

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

**Screenshot 3 (Bottom):** Shows the "Hello\_World\_App.java" file open in the editor. The code is identical to the one in Screenshot 2. The "Output - Run (Hello\_World\_App)" panel shows the build process again: "Recompiling 1 source file with javac [debug release 24] to target\classes", "Compiling 1 source file with javac [debug release 24] to target\classes", and "exec:java: Default-Cls # Hello\_World\_App -> Hello\_World". The status bar at the bottom indicates "BUILD SUCCESS".

9/3/2025      OOP with Java      8

# What is a Class?

- Class is a template to create objects having similar properties and behavior.
- Or we can say that a class is a blueprint for objects.
- Structural template that we use to keep our objects, methods, variables in an order.
- Also we can say it is a representation of the object.

```
access_modifier class<class_name> {  
    data member;  
    method;  
    constructor;  
    nested class;  
    interface  
};  
}
```

# Class Scope

```
class Students {  
  
    // Class-level variable (has class scope)  
    String name;  
  
    // Constructor  
    public Students(String studentName) {  
        name = studentName;  
    }  
  
    // Method 1  
    public void displayName() {  
        System.out.println("Student Name: " + name);  
    }  
  
    // Method 2  
    public void greet() {  
        System.out.println("Hello, " + name + "!");  
    }  
  
    // Main method  
    public static void main(String[] args) {  
        Students s1 = new Students("Nimesha");  
        s1.displayName(); // Accessing class scope variable  
        s1.greet();      // Accessing class scope variable  
    }  
}
```

Access modifier — means the class or method is **visible to all** (any class can access it)

Keyword used to **declare a class** (a blueprint for objects)

Name of the class. By convention, should start with a **Capital Letter**

Braces - enclose the **body of the class or method**

```
1 package com.mycompany.hello_world_app;
2
3 public class Hello_World_App {
4
5     public static void main(String[] args) {
6         System.out.println("Hello World!");
7     }
8 }
9
```

Print statement:  
Print text to the console (no newline)

Means this method belongs to the **class itself**, not an object

Return type — means this method **does not return any value**

The **entry point** of any Java application (method that runs first)

(String [] arg) : Accept **command-line arguments** (an array of Strings named arg)

# Variables

- **Definition:** A container that holds a value which can change during program execution.
- **Components:**
  - Data type - Defines the kind of data stored (e.g., int, String, float).
  - Variable name - A unique identifier following Java naming rules (camelCase).
  - Value - The actual data assigned to the variable.

**Int age = 20;**

—  
Data  
Type

—  
Variable\_name

—  
Value

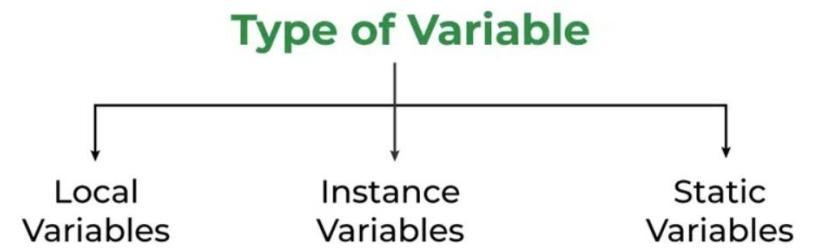
20

Reserved Memory for Variable

RAM

# Variables

```
class VariableExample {  
  
    // Instance variable  
    int instanceVar = 10;  
  
    // Static variable  
    static int staticVar = 20;  
  
    public void showVariables() {  
        // Local variable  
        int localVar = 5;  
  
        System.out.println("Local Variable: " + localVar);      // 5  
        System.out.println("Instance Variable: " + instanceVar); // 10  
        System.out.println("Static Variable: " + staticVar);     // 20  
    }  
  
    public static void main(String[] args) {  
        VariableExample obj = new VariableExample();  
        obj.showVariables();  
    }  
}
```



# Instance vs Static Variables

| Feature                | Instance Variable                                     | Static Variable  |
|------------------------|---|--|
| Copies per object      | Each object has its own copy                          | Only one copy per class (shared by all objects)          |
| Effect of modification | Changes affect only the object                        | Changes affect all objects                               |
| Access method          | Accessed using object reference                       | Accessed using class name                                |
| Lifetime               | Created when object is created, destroyed with object | Created when program starts, destroyed when program ends |
| Syntax example         | <code>int a;</code>                                   | <code>static int a;</code>                               |

# What is an Object?

- Basic unit of Object-Oriented Programming and represents real-life entities.
- An object is an instance of a class that are created to use the attributes and methods of a class.
- An object consists of:
  - State : represented by attributes. Reflect the properties of an object (*what you have*)
  - Behavior : methods of an object. (*what can you do*)
  - Identity : unique name (*who you are*).



# Constructor

- Special block of code that is called when an object is created
- **Key points of Constructors:**
  - Same Name as the Class
  - No need a return type : **purpose is initialize the object, not to return a value**
  - Automatically called on object creation

```
class Bick {  
  
    //create the constructor  
    Bick(){  
        System.out.println("I am the one who work first");  
    }  
  
    //Main method  
    public static void main(String[] args) {  
  
        //Create the Object  
        Bick obj_bick = new Bick();  
    }  
}
```

# Constructor

- **Types of Constructors in Java**
  - Default Constructor
  - Parameterized Constructor

```
// Java Program to demonstrate
// Default Constructor
// Driver class
class Car{

    // Default Constructor
    Car() {
        System.out.println("Default constructor");

    }

    // Driver function
    public static void main(String[] args)
    {
        Car hello = new Car();
    }
}
```

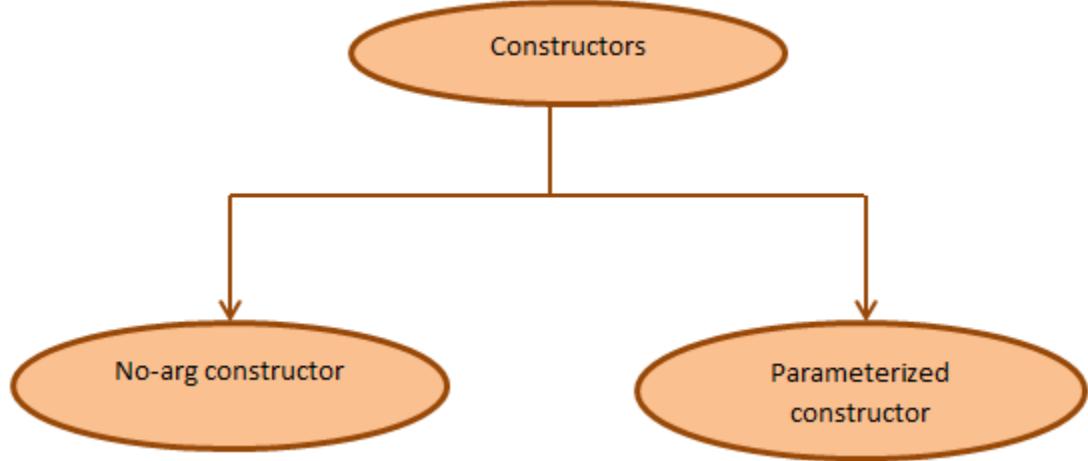
```
class Student {

    // data members of the class
    String name;
    int id;

    Student(String name, int id) {
        this.name = name;
        this.id = id;
    }
}

class GFG
{
    public static void main(String[] args)
    {
        // This would invoke the parameterized constructor
        Student student1 = new Student("Anura", 68);
        System.out.println("Student Name: " + student1.name
                           + " and Student Id: " + student1.id);
    }
}
```

OOP

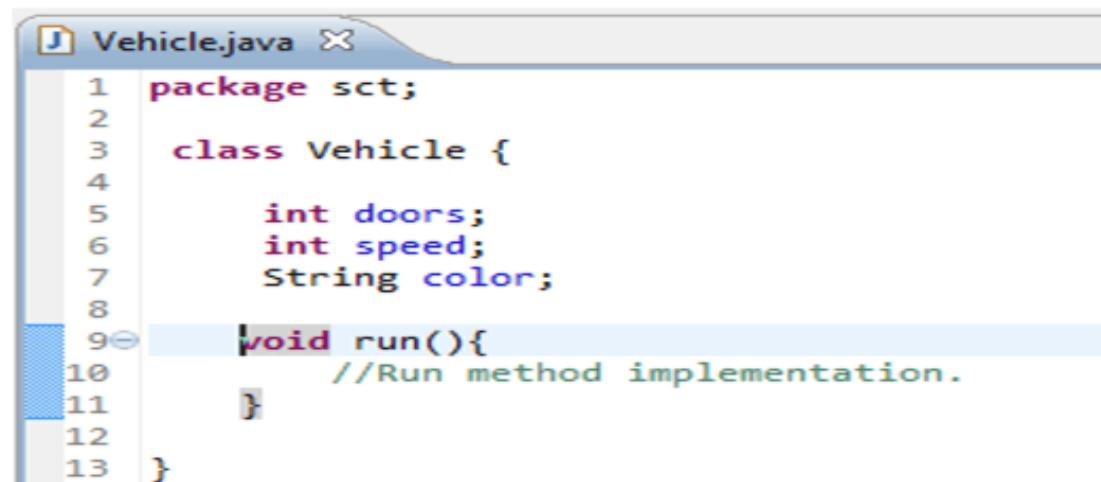


# Object Generation

```
class Animal {  
    // Instance variable  
    String type = "Dog";  
  
    // Method  
    void makeSound() {  
        System.out.println("The " + type + " says: Woof!");  
    }  
  
    public static void main(String[] args) {  
        // Object creation  
        Animal myAnimal = new Animal();  
  
        // Calling method using the object  
        myAnimal.makeSound();  
    }  
}
```

# Methods

- Blocks of code that perform a specific task
- It shows **behaviors** that **change** from each others
- Allows us to reuse code, improving both efficiency and organization
- All **methods in Java** must belong to a **class**



A screenshot of a Java code editor showing a file named "Vehicle.java". The code defines a class "Vehicle" with three instance variables: "doors", "speed", and "color". It also contains a method "run()" with a comment explaining its purpose. The code is numbered from 1 to 13.

```
1 package sct;
2
3 class Vehicle {
4
5     int doors;
6     int speed;
7     String color;
8
9     void run(){
10         //Run method implementation.
11     }
12
13 }
```

# Methods

- Pre Define Methods:
  - Already defined in the Java class libraries

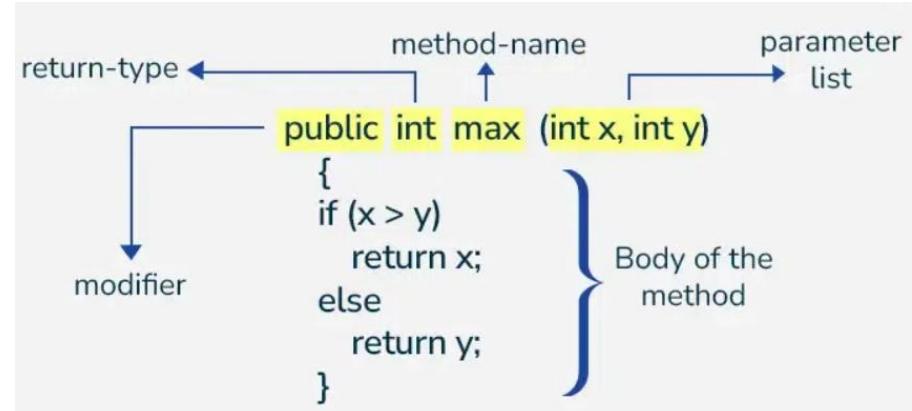
```
Math.random()    // returns random value  
Math.PI()       // return pi value
```

- User Define Methods:
  - Written by the user or programmer

```
public void printMessage() {  
    System.out.println("Hello, Java!");  
}
```

# Methods

- There are points that we need to consider to create a method
  - Modifier : **access level** [based on the requirement]
  - Return type : type of value **returned**, or **void** if **no value is returned**
  - Method name : **must**
  - Parameters : list of **input parameters** inside parentheses ()
  - Method body : **scope {}**



# All Together

```
// A simple class to introduce Class, Constructor, Methods, Variables, and Objects
public class Student {
    // Attributes (variables)
    String name;
    int age;
    // Constructor (to initialize variables)
    public Student(String n, int a) {
        name = n;
        age = a;
    }
    // Method 1
    public void displayInfo() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
    // Method 2
    public void study() {
        System.out.println(name + " is studying.");
    }
    // Main method
    public static void main(String[] args) {
        // Create an object of Student
        Student s1 = new Student("Anura", 22);

        // Call methods using the object
        s1.displayInfo();    // shows student details
        s1.study();         // shows action
    }
}
```