
# Cambodian University of Specialties

## Faculty of Science and Technology

## Java OOP

# Java Class

- In Java, classes and objects are basic concepts of Object Oriented Programming (OOPs) that are used to represent real-world concepts and entities. The class represents a group of objects having similar properties and behavior. For example, the animal type **Dog** is a class while a particular dog named **Tommy** is an object of the **Dog** class.

# Java Class

- Properties of Java Classes
  - Class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
  - Class is a group of variables of different data types and a group of methods.

# Java Class

- Properties of Java Classes
  - A Class in Java can contain:
    - ❖ Data member
    - ❖ Method
    - ❖ Constructor

# Java Class

- **Class Declaration in Java**

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

# Java Class

- **Class Declaration in Java**

```
public class Person
{
    // Data Members or Fields
    private String name;
    private int age;
}
```

# Java Class

- **Class Declaration in Java**

```
public class Person
{
    // Fields
    private String name;
    private int age;
    // Constructor
    public Person(String name, int age)
    {
        this.name = name;
        this.age = age;
    }
}
```

# Java Class

- **Class Declaration in Java**

```
public class Person
{
    // Fields
    private String name;
    private int age;
    // Getter and Setter methods
    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }
}
```

# Java Class

- **Class Declaration in Java**

```
public class Person
{
    // Fields
    private String name;
    private int age;
    // A method to display person details
    public void display()
    {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}
```

# Java Class

- **Access Modifier**
  - In Java, access modifiers are keywords that set the accessibility of classes, methods, and other members. They control where these members can be accessed from. Here are the four main access modifiers in Java:

# Java Class

- **Access Modifier**
  - **public**: The member is accessible from any other class.
  - **private**: The member is accessible only within the class it is declared.
  - **protected:** The member is accessible within the same package and by subclasses.
  - **default (package-private)**: If no access modifier is specified, the member is accessible only within the same package.

# Java Class

- **Access Modifier**

| Modifier | Class | Package | Subclass | World |
|---|---|---|---|---|
| public | Yes | Yes | Yes | Yes |
| protected | Yes | Yes | Yes | No |
| no modifier | Yes | Yes | No | No |
| private | Yes | No | No | No |

# Java Class

- **Access Modifier**

```java
// Java Program for class example
class Student
{
    // data member (also instance variable)
    int id;
    // data member (also instance variable)
    String name;

    public static void main(String args[])
    {
        // creating an object of
        // Student
        Student s1 = new Student();
        System.out.println(s1.id);
        System.out.println(s1.name);
    }
}
```

# Java Class

- **Class Declaration in Java**

```
public class Person
{
    // Fields (or instance variables)
    private String name;
    private int age;
}
```

# Java Class (Data Members)

- In Java, data members or fields are variables that are declared within a class. They represent the state or attributes of an object created from the class. Fields can be of any data type, including primitive types (like int, float, boolean) and reference types (like objects and arrays).

# Java Class (Data Members)

```java
public class Person
{
    // Fields
    private String name;
    private int age;
}
```

# Java Class (Data Members)

- name and age are fields of the Person class. They store the name and age of a person, respectively.
- Fields are typically declared as private to encapsulate the data and protect it from unauthorized access. Access to these fields is provided through public getter and setter methods.

# Java Class (Methods)

- In Java, methods are blocks of code that perform a specific task. They are used to define the behavior of objects created from a class. Methods can take input in the form of parameters, perform operations, and return a result. They help in organizing code into reusable and manageable sections.

# Java Class (Methods)

```java
public class Example {
    public static void main(String[] args) {
        printMessage();
    }

    // Non-return type method
    public static void printMessage() {
        System.out.println("Hello, World!");
    }
}
```

# Java Class (Methods)

```java
public class Calculator {
    public static void main(String[] args) {
        Calculator calc = new Calculator();

        int sum = calc.add(5, 3);
        System.out.println("Sum: " + sum); // Output: Sum: 8

        double area = calc.calculateCircleArea(7);
        System.out.println("Area: " + area); // Output: Area: 153.938040025
    }

    // Return type method that returns an int
    public int add(int a, int b) {
        return a + b;
    }

    // Return type method that returns a double
    public double calculateCircleArea(double radius) {
        return Math.PI * radius * radius;
    }
}
```

# Java Class (setter and getter Methods)

- Setter and getter methods in Java are used to access and update the values of private variables. They are part of the encapsulation principle in object-oriented programming, which helps to protect the data from unauthorized access and modification.

# Java Class (setter and getter Methods)

```
public class Person
  {
      // Private variable
      private String name;
      // Getter method for name
      public String getName()
      {
          return name;
      }
      // Setter method for name
      public void setName(String name) {
          this.name = name;
      }
  }
```

# Java Class (Constructor)

- What is java constructor
  - In Java, a constructor is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes. Here are some key points about constructors:
  - Name: The constructor has the same name as the class.
  - No Return Type: Constructors do not have a return type, not even void.
  - Types: There are two types of constructors:
    - Default Constructor: A constructor with no parameters.
    - Parameterized Constructor: A constructor that takes one or more parameters.

# Java Class (Constructor)

```java
public class Car
{
    private String model;
    private int year;
    // Default constructor
    public Car() {
        model = "Unknown";
        year = 0;
    }
    // Parameterized constructor
    public Car(String model, int year)
    {
        this.model = model;
        this.year = year;
    }
    public void displayInfo()
    {
        System.out.println("Model: " + model + ", Year: " + year);
    }
}
```

# Java Class (Constructor)

- The Car class has two constructors: a default constructor and a parameterized constructor.
- The default constructor initializes the model and year attributes to default values.
- The parameterized constructor initializes the model and year attributes to the values provided as arguments.
- Constructors are essential for creating and initializing objects in Java.

# Java Class (Object)

- In Java, an object is an instance of a class. Objects are the fundamental building blocks of object-oriented programming (OOP). They encapsulate data and behavior, allowing you to model real-world entities and interactions in your programs.

# Java Class (Object)

```java
public class Dog {
    // Attributes (fields)
    String name;
    int age;
    // Method (behavior)
    void bark()
    {
        System.out.println(name + " is barking!");
    }
    public static void main(String[] args) {

        // Creating an object of the Dog class
        Dog myDog = new Dog();

        // Setting attributes
        myDog.name = "Buddy";
        myDog.age = 3;

        // Calling a method
        myDog.bark(); // Output: Buddy is barking!
    }
}
```

# Java Class (Object)

- The Dog class defines the attributes (name and age) and behavior (bark method) of a dog.
- The myDog object is an instance of the Dog class.
- The attributes of the myDog object are set to "Buddy" and 3.
- The bark method is called on the myDog object, which prints "Buddy is barking!" to the console.
- Objects allow you to create multiple instances of a class, each with its own set of attributes and behaviors. This makes it easier to manage and organize your code, especially in larger programs.