Introduction to Java Object-Oriented Programming

A Comprehensive Guide to Classes, Objects, Inheritance, Polymorphism, Encapsulation, and Abstraction

Agenda

Classes and Objects

Inheritance

Polymorphism

Encapsulation and Abstraction

Lab: Inheritance and Polymorphism

Classes and Objects

Definition:

Class: A blueprint for creating objects. It defines properties and behaviors.

Object: An instance of a class.

Key Concepts:

Fields (attributes)

Methods (functions)

Constructors

```java
class Car {
    String brand;
    int speed;

    void displayInfo() {
        System.out.println("Brand: " + brand);
        System.out.println("Speed: " + speed);
    }
}

public class Main {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.brand = "Toyota";
        myCar.speed = 120;
        myCar.displayInfo();
    }
```

Inheritance

Definition:

Inheritance allows a class (child) to acquire the properties and behaviors of another class (parent).

Key Concepts:

extends **Keyword**: Used to inherit a class.

Superclass (Parent Class) and Subclass (Child Class)

Benefits:

Code Reusability

Method Overriding

Example:

```
class Animal {
    void eat() {
        System.out.println("This animal eats food.");
    }
}
class Dog extends Animal {
    void bark() {
        System.out.println("The dog barks.");
    }
}
public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();
        dog.bark();
    }
}
```

Example:

Polymorphism

Definition:

Polymorphism allows one interface to be used for a general class of actions.

Key Concepts:

Compile-Time Polymorphism (Method Overloading)

Runtime Polymorphism (Method Overriding)

Example: Compile-Time Polymorphism:

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }
}
```

Example: Runtime Polymorphism

```
class Animal {
    void sound() {
        System.out.println("This animal makes a sound.");
    }
}
class Cat extends Animal {
    void sound() {
        System.out.println("The cat meows.");
    }
}
public class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Cat(); // Upcasting
        myAnimal.sound();
    }
}
```

Encapsulation and Abstraction

Encapsulation

Definition:

Encapsulation binds data and methods together while restricting access to some components.

Key Concepts:

Access Modifiers: private, public, protected

Getters and Setters

Example

```
class Person {
    private String name;

    public String getName() {
        return name;
    }

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

Abstraction

Definition:

Hiding implementation details and showing only essential features.

Key Concepts:

Abstract Classes

Interfaces

Example

```
abstract class Animal {
    abstract void makeSound();
}

class Cow extends Animal {
    void makeSound() {
        System.out.println("Cow moos.");
    }
}
```

```
interface Vehicle {
    void start();
}

class Car implements Vehicle {
    public void start() {
        System.out.println("Car is starting.");
    }
}
```

Lab: Inheritance and Polymorphism

Objective:

Understand how to implement inheritance and polymorphism in Java.

Task:

Create a base class Shape with a method area().

Derive two classes Circle Triangle and Rectangle from Shape. Override the area() method in both classes.

Demonstrate runtime polymorphism by calling the area() method using a Shape reference

Solution: