HYDERABAD INSTITUTE OF ARTS, SCIENCE, AND TECHNOLOGY

Object Oriented Programming lab-10

Instructor: Miss Ayesha Eman

Date: 13/10/2025

Title:

Abstract Classes and Interfaces

Objectives

- To understand abstraction and its role in OOP.
- To implement abstract classes and interfaces in real-world scenarios.
- To explore multiple inheritance through interfaces.
- To design flexible and scalable systems using abstraction.

Definition

Abstraction is one of the core principles of Object-Oriented Programming (OOP). It hides implementation details and exposes only the necessary functionality. In Java, abstraction is achieved through abstract classes and interfaces.

An abstract class serves as a blueprint that defines structure but may include partial implementation. Subclasses extending it must implement the abstract methods, allowing shared code and enforced design patterns.

An interface provides complete abstraction. It defines method signatures that implementing classes must fulfill, allowing Java to support multiple inheritance and ensuring a consistent contract for behavior.

Example 1: Abstract Class - Employee Hierarchy

```
public class Main {
    public static void main(String[] args) {
        Employee e1 = new Developer("Fiza", 90000);
        Employee e2 = new Manager("Arif", 120000);
        e1.displayDetails();
        System.out.println("Bonus: " + e1.calculateBonus());
        e2.displayDetails();
        System.out.println("Bonus: " + e2.calculateBonus());
    }
}
abstract class Employee {
    String name;
    double salary;
    // Constructor
    Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }
    // Abstract method
    abstract double calculateBonus();
    // Concrete method
    void displayDetails() {
        System.out.println("Employee Name: " + name);
        System.out.println("Base Salary: " + salary);
    }
}
```

```
class Developer extends Employee {

    Developer(String name, double salary) {

        super(name, salary);

    }



    double calculateBonus() {

        return salary * 0.15;

    }

}

class Manager extends Employee {

    Manager(String name, double salary) {

        super(name, salary);

    }

    double calculateBonus() {

        return salary * 0.25;

    }

}
```

Example 2: Interface - Smart Devices

```
interface SmartDevice {
    void connect();
    void batteryStatus();
}

class Phone implements SmartDevice {
    public void connect() {
        System.out.println("Phone connected to Wi-Fi.");
    }
    public void batteryStatus() {
        System.out.println("Phone battery at 85%.");
    }
}

class SmartWatch implements SmartDevice {
    public void connect() {
        System.out.println("SmartWatch connected via Bluetooth.");
    }
    public void batteryStatus() {
        System.out.println("SmartWatch battery at 60%.");
```

```
  }
}

public class TestDevice {
  public static void main(String[] args) {
    SmartDevice d1 = new Phone();
    SmartDevice d2 = new SmartWatch();
    d1.connect();
    d1.batteryStatus();
    d2.connect();
    d2.batteryStatus();
  }
}
```

Output

```
Phone connected to Wi-Fi.
Phone battery at 85%.
SmartWatch connected via Bluetooth.
SmartWatch battery at 60%.
```

Example 3: Multiple Inheritance Using Interfaces - Hybrid Car

```
interface Engine {
  void startEngine();
}

interface Battery {
  void chargeBattery();
}

class HybridCar implements Engine, Battery {
  public void startEngine() {
    System.out.println("Hybrid engine started with electric support.");
  }
  public void chargeBattery() {
    System.out.println("Hybrid battery charging via regenerative braking.");
  }
}

public class HybridTest {
  public static void main(String[] args) {
    HybridCar car = new HybridCar();
    car.startEngine();
    car.chargeBattery();
  }
}
```

Output

```
Hybrid engine started with electric support.
Hybrid battery charging via regenerative braking.
```

Discussion Questions

1. What is abstraction, and how is it implemented in Java?
2. How does an abstract class differ from an interface?
3. Why can interfaces allow multiple inheritance but classes cannot?
4. In what scenarios should abstract classes be preferred over interfaces?
5. How does abstraction enhance modularity and code flexibility?