LAB PROBLEM 1: Abstract Fruit and Edible Interface (Any Four)

Topic: Abstract Class with Interface Implementation

Problem Statement:

Create an abstract class Fruit with protected fields color and taste. Add an abstract method showDetails().

Create an interface Edible with method nutrientsInfo().

Create a class Apple that extends Fruit and implements Edible, adding a variety field.

Hints:

● Use abstract for parent class.

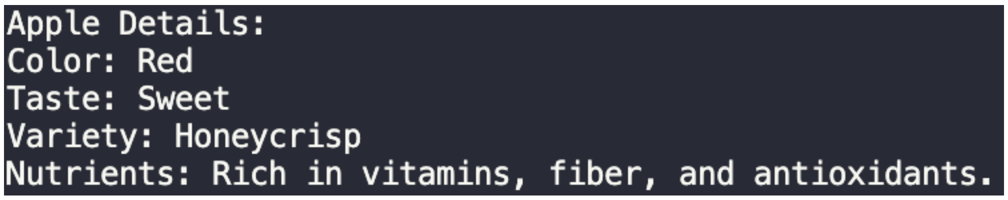
● Use interface for common behavior.

● Implement both abstract and interface methods.

```
// Abstract class Fruit
abstract class Fruit {
   protected String color;
   protected String taste;

   // Constructor
   Fruit(String color, String taste) {
      this.color = color;
      this.taste = taste;
   }

   // Abstract method
   public abstract void showDetails();
}
```

```
// Interface Edible
interface Edible {
  void nutrientsInfo();
}

// Apple class extends Fruit and implements Edible
class Apple extends Fruit implements Edible {
  private String variety;

  // Constructor
  Apple(String color, String taste, String variety) {
    super(color, taste);
    this.variety = variety;
  }

  // Implement abstract method
  @Override
  public void showDetails() {
    System.out.println("Apple Details:");
    System.out.println("Color: " + color);
    System.out.println("Taste: " + taste);
    System.out.println("Variety: " + variety);
  }

  // Implement interface method
  @Override
  public void nutrientsInfo() {
    System.out.println("Nutrients: Rich in vitamins, fiber, and antioxidants.");
  }
}

// Main class to test
public class FruitTest {
  public static void main(String[] args) {
    Apple myApple = new Apple("Red", "Sweet", "Honeycrisp");

    // Call methods
    myApple.showDetails();
    myApple.nutrientsInfo();
  }
}
```

OUTPUT:-



```
Apple Details:
Color: Red
Taste: Sweet
Variety: Honeycrisp
Nutrients: Rich in vitamins, fiber, and antioxidants.
```

LAB PROBLEM 2: Abstract Shape and Drawable Interface

Topic: Abstract Class and Interface in Geometry

Problem Statement:

Create an abstract class Shape with fields area and perimeter. Add abstract methods calculateArea() and calculatePerimeter().

Create an interface Drawable with method draw().

Create a class Circle extending Shape and implementing Drawable.

Hints:

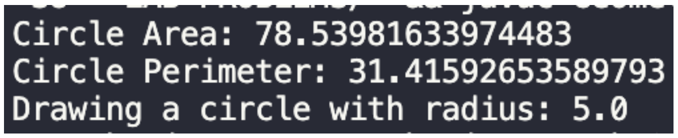
● Abstract methods must be overridden in child class.

● Use interface to add extra behavior.

```
// Abstract class Shape
abstract class Shape {
   protected double area;
   protected double perimeter;


   // Abstract methods
   public abstract void calculateArea();
   public abstract void calculatePerimeter();
}


// Interface Drawable
interface Drawable {
   void draw();
}
```

```
// Circle class extends Shape and implements Drawable
class Circle extends Shape implements Drawable {
  private double radius;

  // Constructor
  Circle(double radius) {
    this.radius = radius;
  }

  // Implement abstract methods
  @Override
  public void calculateArea() {
    area = Math.PI * radius * radius;
    System.out.println("Circle Area: " + area);
  }

  @Override
  public void calculatePerimeter() {
    perimeter = 2 * Math.PI * radius;
    System.out.println("Circle Perimeter: " + perimeter);
  }

  // Implement interface method
  @Override
  public void draw() {
    System.out.println("Drawing a circle with radius: " + radius);
    }
}

// Main class to test
public class GeometryTest {
    public static void main(String[] args) {
        Circle myCircle = new Circle(5);

        // Call methods
        myCircle.calculateArea();
        myCircle.calculatePerimeter();
        myCircle.draw();
    }
}
```

OUTPUT:-



```
Circle Area: 78.53981633974483
Circle Perimeter: 31.41592653589793
Drawing a circle with radius: 5.0
```

LAB PROBLEM 3: Abstract Vehicle and Maintainable Interface

Topic: Abstract Class and Interface in Transport System

Problem Statement:

Create an abstract class Vehicle with protected fields speed and fuelType. Add an abstract method startEngine().

Create an interface Maintainable with method serviceInfo().

Create a class Car that extends Vehicle and implements Maintainable.

Hints:

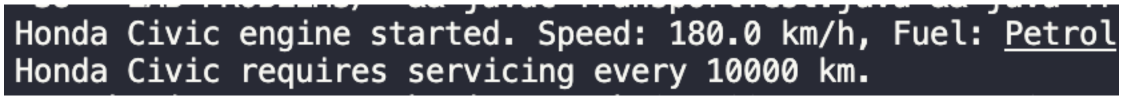
● Use extends and implements together.

● Provide concrete implementations for abstract and interface methods.

```
// Abstract class Vehicle
abstract class Vehicle {
    protected double speed;
    protected String fuelType;

    // Constructor
    Vehicle(double speed, String fuelType) {
        this.speed = speed;
        this.fuelType = fuelType;
    }

    // Abstract method
    public abstract void startEngine();
}
```

```
// Interface Maintainable
interface Maintainable {
  void serviceInfo();
}

// Car class extends Vehicle and implements Maintainable
class Car extends Vehicle implements Maintainable {
  private String model;

  // Constructor
  Car(double speed, String fuelType, String model) {
    super(speed, fuelType);
    this.model = model;
  }

  // Implement abstract method
  @Override
  public void startEngine() {
    System.out.println(model + " engine started. Speed: " + speed + " km/h, Fuel: " +
fuelType);
  }

  // Implement interface method
  @Override
  public void serviceInfo() {
    System.out.println(model + " requires servicing every 10000 km.");
    }
}


// Main class to test
public class TransportTest {
    public static void main(String[] args) {
        Car myCar = new Car(180, "Petrol", "Honda Civic");


        // Call methods
        myCar.startEngine();
        myCar.serviceInfo();
    }
}
```

OUTPUT:-



```
Honda Civic engine started. Speed: 180.0 km/h, Fuel: Petrol
Honda Civic requires servicing every 10000 km.
```

LAB PROBLEM 4: Abstract Employee and Payable Interface

Topic: Abstract Class with Interface for Payroll System

Problem Statement:

Create an abstract class Employee with fields name and salary. Add abstract method calculateBonus().

Create an interface Payable with method generatePaySlip().

Create a class Manager that extends Employee and implements Payable.

Hints:

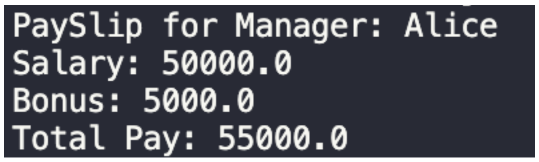
- Use abstract method for bonus calculation.
- Interface method should handle pay slip generation.

```
// Abstract class Employee
abstract class Employee {
  protected String name;
  protected double salary;

  // Constructor
  Employee(String name, double salary) {
    this.name = name;
    this.salary = salary;
  }

  // Abstract method
  public abstract double calculateBonus();
}
```

```
// Interface Payable
interface Payable {
  void generatePaySlip();
}

// Manager class extends Employee and implements Payable
class Manager extends Employee implements Payable {
  private double bonusPercentage;

  // Constructor
  Manager(String name, double salary, double bonusPercentage) {
    super(name, salary);
    this.bonusPercentage = bonusPercentage;
  }

  // Implement abstract method
  @Override
  public double calculateBonus() {
    double bonus = salary * bonusPercentage / 100;
    return bonus;
  }

  // Implement interface method
  @Override
  public void generatePaySlip() {
    double bonus = calculateBonus();
```

```
        double totalPay = salary + bonus;

        System.out.println("PaySlip for Manager: " + name);

        System.out.println("Salary: " + salary);

        System.out.println("Bonus: " + bonus);

        System.out.println("Total Pay: " + totalPay);

    }

}


// Main class to test

public class PayrollTest {

    public static void main(String[] args) {

        Manager manager = new Manager("Alice", 50000, 10);


        // Call methods

        manager.generatePaySlip();

    }

}
```

OUTPUT:-



```
PaySlip for Manager: Alice
Salary: 50000.0
Bonus: 5000.0
Total Pay: 55000.0
```