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.

```
public abstract class Fruit {
    protected String color;
    protected String taste;
    public Fruit(String color, String taste){
        this.color = color;
        this.taste = taste;
    }
    public abstract void showDetails();
}
```

```
public interface Edible {
```

```
    void nutrientsInfo();
}


public class Apple extends Fruit implements Edible {
    private String variety;
    public Apple(String color, String taste, String variety){
        super(color, taste);
        this.variety = variety;
    }
    public void showDetails(){
        System.out.println("Apple variety: " + variety);
        System.out.println("Color: " + color);
        System.out.println("Taste: " + taste);
    }
    public void nutrientsInfo(){
        System.out.println("Rich in fiber and vitamin C.");
    }
    public static void main(String[] args){
        Apple a = new Apple("Red", "Sweet", "Honeycrisp");
        a.showDetails();
        a.nutrientsInfo();
    }
}
```

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.

```
public abstract class Shape {
    protected double area;
    protected double perimeter;
    public abstract void calculateArea();
    public abstract void calculatePerimeter();
}

public interface Drawable {
    void draw();
}

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

```
    public Circle(double radius){
        this.radius = radius;
    }
    public void calculateArea(){
        area = Math.PI * radius * radius;
    }
    public void calculatePerimeter(){
        perimeter = 2 * Math.PI * radius;
    }
    public void draw(){
        System.out.println("Drawing a circle with radius: " + radius);
    }
    public static void main(String[] args){
        Circle c = new Circle(5.0);
        c.calculateArea();
        c.calculatePerimeter();
        c.draw();
        System.out.println("Area: " + c.area);
        System.out.println("Perimeter: " + c.perimeter);
    }
}

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

```
    public Circle(double radius){
        this.radius = radius;
    }
    public void calculateArea(){
        area = Math.PI * radius * radius;
    }
    public void calculatePerimeter(){
        perimeter = 2 * Math.PI * radius;
    }
    public void draw(){
        System.out.println("Drawing a circle with radius: " + radius);
    }
    public static void main(String[] args){
        Circle c = new Circle(5.0);
        c.calculateArea();
        c.calculatePerimeter();
        c.draw();
        System.out.println("Area: " + c.area);
        System.out.println("Perimeter: " + c.perimeter);
    }
}
```

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.

```
public abstract class Vehicle {

    protected double speed;

    protected String fuelType;

    public Vehicle(double speed, String fuelType){

        this.speed = speed;
```

```
        this.fuelType = fuelType;
    }
    public abstract void startEngine();
}

public interface Maintainable {
    void serviceInfo();
}

public class Car extends Vehicle implements Maintainable {
    private String model;
    public Car(double speed, String fuelType, String model){
        super(speed, fuelType);
        this.model = model;
    }
```

```
    public void startEngine(){

        System.out.println("Starting engine of " + model + ".");

    }

    public void serviceInfo(){

        System.out.println("Service recommended every 10000 km for " +
model + ".");

    }

    public static void main(String[] args){

        Car car = new Car(180.0, "Petrol", "SedanX");

        car.startEngine();

        car.serviceInfo();

        System.out.println("Top speed: " + car.speed);

        System.out.println("Fuel type: " + car.fuelType);

    }

}
```

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.

```
public abstract class Employee {
    protected String name;
    protected double salary;
    public Employee(String name, double salary){
        this.name = name;
        this.salary = salary;
    }
    public abstract double calculateBonus();
}
```

```
public interface Payable {
    void generatePaySlip();
}

public class Manager extends Employee implements Payable {
    private String department;
    public Manager(String name, double salary, String department){
        super(name, salary);
        this.department = department;
    }
    public double calculateBonus(){
        return salary * 0.15;
    }
    public void generatePaySlip(){
        System.out.println("Pay slip for: " + name);
        System.out.println("Department: " + department);
        System.out.println("Basic salary: " + salary);
        System.out.println("Bonus: " + calculateBonus());
        System.out.println("Net pay: " + (salary + calculateBonus()));
    }
    public static void main(String[] args){
        Manager m = new Manager("Riya", 80000, "Operations");
        m.generatePaySlip();
```

```
    }
}
```


LAB PROBLEM 5: Abstract Animal and Soundable Interface

Topic: Abstract Class and Interface in Zoology

Problem Statement:

Create an abstract class `Animal` with fields `name` and `habitat`. Add an abstract method `eat()`.

Create an interface `Soundable` with method `makeSound()`.

Create a class `Dog` that extends `Animal` and implements `Soundable`. **Hints:**

- Abstract method represents common but incomplete behavior.
- Interface enforces sound-making behavior across animals.

LAB PROBLEM 6: Abstract Device and Connectable Interface

Topic: Abstract Class and Interface in Electronics

Problem Statement:

Create an abstract class `Device` with fields `brand` and `model`. Add an abstract method `powerOn()`.

Create an interface `Connectable` with method `connect()`.

Create a class `Smartphone` that extends `Device` and implements `Connectable`. **Hints:**

- Abstract class handles general device structure.
- Interface enforces connectivity feature.