

Ques.1

Create an **abstract class** `Shape` that represents different geometric shapes. This class should contain:

1. A **protected attribute** `shapeName` to store the name of the shape.
2. A **constructor** that initializes `shapeName`.
3. A **method** `getShapeName()` that returns the name of the shape.
4. An **abstract method** `calculateArea()` that will be implemented by subclasses.

Next, create two concrete classes, `Circle` and `Rectangle`, that extend `Shape` and implement the `calculateArea()` method:

- The `Circle` class should have a **private attribute** `radius`, a constructor to initialize it, and implement `calculateArea()` using the formula:

$$\text{Area} = \pi \times \text{radius}^2$$

- The `Rectangle` class should have **private attributes** `length` and `width`, a constructor to initialize them, and implement `calculateArea()` using the formula:

$$\text{Area} = \text{length} \times \text{width}$$

Finally, write a **test program** in the `main` method that:

1. Creates an object of `Circle` with a radius of `5.0` and displays the area.
2. Creates an object of `Rectangle` with a length of `4.0` and width of `6.0` and displays the area.
3. Uses **polymorphism** by referring to objects of `Circle` and `Rectangle` using `Shape` references.

Note: Implement the concepts of **abstraction, method overriding, polymorphism, and encapsulation in Java OOP concepts.**

Ques.1 - Abstract Class and Polymorphism

Sol:

```
// Abstract class Shape
abstract class Shape {
    protected String shapeName;

    public Shape(String shapeName) {
        this.shapeName = shapeName;
    }
}
```

```

    }

    public String getShapeName() {
        return shapeName;
    }

    // Abstract method for calculating area
    public abstract double calculateArea();
}

// Circle class extending Shape
class Circle extends Shape {
    private double radius;

    public Circle(double radius) {
        super("Circle");
        this.radius = radius;
    }

    @Override
    public double calculateArea() {
        return Math.PI * radius * radius;
    }
}

// Rectangle class extending Shape
class Rectangle extends Shape {
    private double length, width;

    public Rectangle(double length, double width) {
        super("Rectangle");
        this.length = length;
        this.width = width;
    }

    @Override
    public double calculateArea() {
        return length * width;
    }
}

// Main class to test polymorphism
public class ShapeTest {
    public static void main(String[] args) {
        // Using polymorphism
        Shape shape1 = new Circle(5.0);
        System.out.printf(shape1.getShapeName() + " Area: %.2f \n", shape1.calculateArea());
    }
}

```

```

        Shape shape2 = new Rectangle(4.0, 6.0);
        System.out.printf(shape2.getShapeName() + " Area: %.2f", shape2.calculateArea());
    }
}

```

Output:

Circle Area: 78.54

Rectangle Area: 24.00

```

// ShapeTest.java
1 // Abstract class Shape
2 abstract class Shape {
3     protected String shapeName;
4
5     public Shape(String shapeName) {
6         this.shapeName = shapeName;
7     }
8
9     public String getShapeName() {
10        return shapeName;
11    }
12
13    // Abstract method for calculating area
14    public abstract double calculateArea();
15 }
16
17 // Circle class extending Shape
18 class Circle extends Shape {
19     private double radius;
20
21     public Circle(double radius) {
22         super("Circle");
23         this.radius = radius;
24     }
25
26     @Override
27     public double calculateArea() {
28         return Math.PI * radius * radius;
29     }
30 }
31
32 // Rectangle class extending Shape
33 class Rectangle extends Shape {
34     private double length, width;
35
36     public Rectangle(double length, double width) {
37         super("Rectangle");
38         this.length = length;
39         this.width = width;
40     }
41
42     @Override
43     public double calculateArea() {
44         return length * width;
45     }
46 }
47
48 // Main class to test polymorphism
49 public class ShapeTest {
50     public static void Main(String[] args) {
51         // Using polymorphism
52         Shape shape1 = new Circle(5.0);
53         System.out.printf(shape1.getShapeName() + " Area: %.2f\n", shape1.calculateArea());
54
55         Shape shape2 = new Rectangle(4.0, 6.0);
56         System.out.printf(shape2.getShapeName() + " Area: %.2f", shape2.calculateArea());
57     }
58 }

```

Output Console:

```

Circle Area: 78.54
Rectangle Area: 24.00
...Program finished with exit code 0
Press ENTER to exit console.

```

Ques.2

Create an **interface** **Photosynthesis**, which represents the ability of plants to absorb sunlight and perform photosynthesis. This interface should contain a method `void absorbSunlight()`; that defines how plants absorb sunlight. Additionally, create another **interface** **Respiration**, which represents the respiration process in plants. This interface should include a method `void releaseOxygen()`; that defines how plants release oxygen.

Next, implement a **concrete class** **Plant** that inherits from both interfaces, demonstrating multiple inheritance. The **Plant** class should have a **private attribute** **plantName** to store the name of the plant, which should be initialized using a

constructor `Plant(String name)`. Additionally, it should have a method `getPlantName()` to return the name of the plant. The class should override the `absorbSunlight()` method to print the message **"Plant is absorbing sunlight for photosynthesis."** and override the `releaseOxygen()` method to print **"Plant is releasing oxygen through respiration."**.

In the main method, test the implementation by creating objects of the `Plant` class for two different plants: "Mango Tree" and "Fern". For each object, display the plant's name, call the `absorbSunlight()` method, and call the `releaseOxygen()` method.

===

Ques.2 - Interface and Multiple Inheritance

Sol:

```
// Photosynthesis Interface
interface Photosynthesis {
    void absorbSunlight();
}
```

```
// Respiration Interface
interface Respiration {
    void releaseOxygen();
}
```

```
// Plant class implementing both interfaces
class Plant implements Photosynthesis, Respiration {
    private String plantName;

    public Plant(String plantName) {
        this.plantName = plantName;
    }

    public String getPlantName() {
        return plantName;
    }

    @Override
    public void absorbSunlight() {
        System.out.println(plantName + " is absorbing sunlight for photosynthesis.");
    }

    @Override
    public void releaseOxygen() {
        System.out.println(plantName + " is releasing oxygen through respiration.");
    }
}
```

```
// Main class to test implementation
public class PlantTest {
    public static void main(String[] args) {
        Plant mangoTree = new Plant("Mango Tree");
        Plant fern = new Plant("Fern");

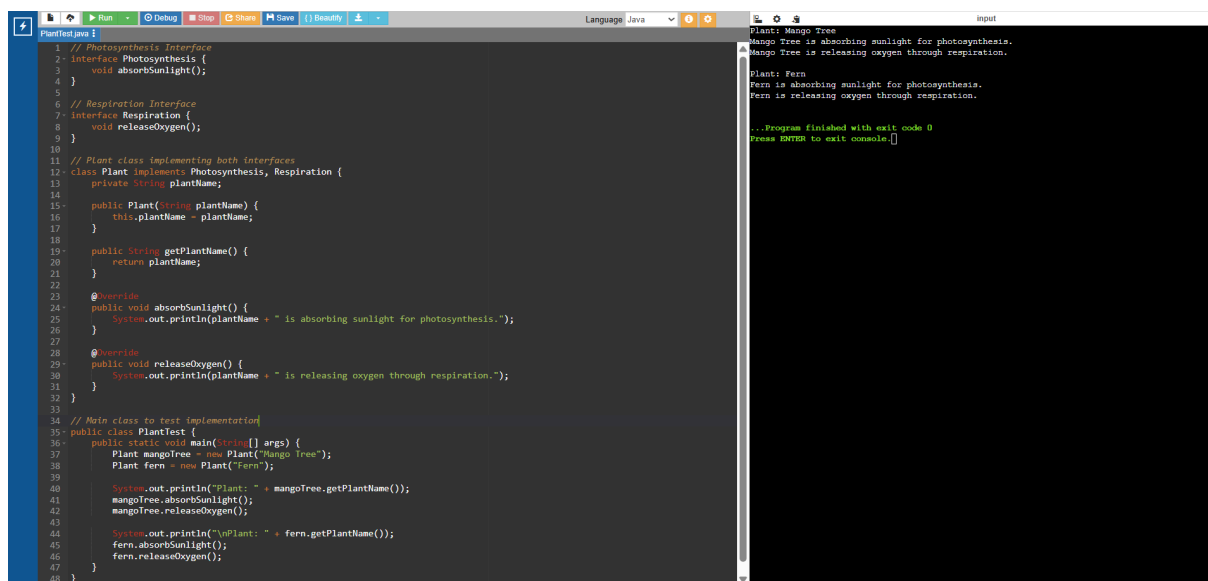
        System.out.println("Plant: " + mangoTree.getPlantName());
        mangoTree.absorbSunlight();
        mangoTree.releaseOxygen();

        System.out.println("\nPlant: " + fern.getPlantName());
        fern.absorbSunlight();
        fern.releaseOxygen();
    }
}
```

Output:

Plant: Mango Tree
Mango Tree is absorbing sunlight for photosynthesis.
Mango Tree is releasing oxygen through respiration.

Plant: Fern
Fern is absorbing sunlight for photosynthesis.
Fern is releasing oxygen through respiration.



The screenshot shows an IDE with the following code in `PlantTest.java`:

```
1 // Photosynthesis Interface
2 interface Photosynthesis {
3     void absorbSunlight();
4 }
5
6 // Respiration Interface
7 interface Respiration {
8     void releaseOxygen();
9 }
10
11 // Plant class implementing both interfaces
12 class Plant implements Photosynthesis, Respiration {
13     private String plantName;
14
15     public Plant(String plantName) {
16         this.plantName = plantName;
17     }
18
19     public String getPlantName() {
20         return plantName;
21     }
22
23     @Override
24     public void absorbSunlight() {
25         System.out.println(plantName + " is absorbing sunlight for photosynthesis.");
26     }
27
28     @Override
29     public void releaseOxygen() {
30         System.out.println(plantName + " is releasing oxygen through respiration.");
31     }
32 }
33
34 // Main class to test implementation
35 public class PlantTest {
36     public static void main(String[] args) {
37         Plant mangoTree = new Plant("Mango Tree");
38         Plant fern = new Plant("Fern");
39
40         System.out.println("Plant: " + mangoTree.getPlantName());
41         mangoTree.absorbSunlight();
42         mangoTree.releaseOxygen();
43
44         System.out.println("\nPlant: " + fern.getPlantName());
45         fern.absorbSunlight();
46         fern.releaseOxygen();
47     }
48 }
```

The console output on the right shows the following:

```
Plant: Mango Tree
Mango Tree is absorbing sunlight for photosynthesis.
Mango Tree is releasing oxygen through respiration.

Plant: Fern
Fern is absorbing sunlight for photosynthesis.
Fern is releasing oxygen through respiration.

...Program finished with exit code 0
Press ENTER to exit console.
```

Ques.3

Java Coding Question (20 Marks)

Implement **Runtime Polymorphism** in Java using a **BMW vehicle hierarchy**.

Create a **parent class BMW** with two methods: `showDetails()`, which prints "This is a BMW vehicle.", and `maxSpeed()`, which prints "Speed varies by model.".

Extend this class into **three subclasses**: `BMWSeries3`, `BMWSeries5`, and `BMWSeries7`, each overriding the methods to display their respective series names and max speeds (240 km/h, 260 km/h, and 300 km/h).

In the main method, create an **array of BMW references**, assign subclass objects, and use a loop to call `showDetails()` and `maxSpeed()`. The program should demonstrate **method overriding, runtime polymorphism, and dynamic method dispatch**.

Ques.3 - Runtime Polymorphism using BMW Hierarchy

Sol:

```
// Parent class BMW
class BMW {
    public void showDetails() {
        System.out.println("This is a BMW vehicle.");
    }

    public void maxSpeed() {
        System.out.println("Speed varies by model.");
    }
}

// Subclasses overriding methods
class BMWSeries3 extends BMW {
    @Override
    public void showDetails() {
        System.out.println("This is a BMW Series 3.");
    }

    @Override
    public void maxSpeed() {
```

```

        System.out.println("Max Speed: 240 km/h");
    }
}

class BMWSeries5 extends BMW {
    @Override
    public void showDetails() {
        System.out.println("This is a BMW Series 5.");
    }

    @Override
    public void maxSpeed() {
        System.out.println("Max Speed: 260 km/h");
    }
}

class BMWSeries7 extends BMW {
    @Override
    public void showDetails() {
        System.out.println("This is a BMW Series 7.");
    }

    @Override
    public void maxSpeed() {
        System.out.println("Max Speed: 300 km/h");
    }
}

// Main class to test runtime polymorphism
public class BMWTest {
    public static void main(String[] args) {
        BMW[] cars = { new BMWSeries3(), new BMWSeries5(), new BMWSeries7() };

        for (BMW car : cars) {
            car.showDetails();
            car.maxSpeed();
            System.out.println();
        }
    }
}

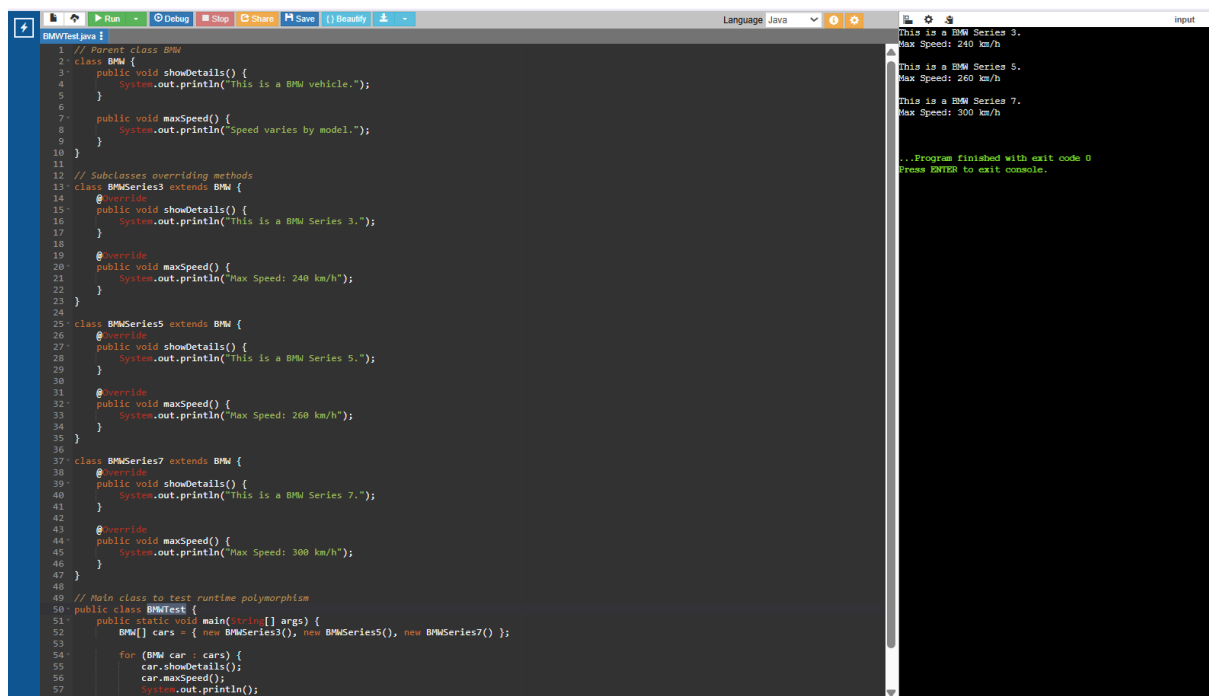
```

Output:

This is a BMW Series 3.
Max Speed: 240 km/h

This is a BMW Series 5.
Max Speed: 260 km/h

This is a BMW Series 7.
Max Speed: 300 km/h



```
1 // Parent class BMW
2 class BMW {
3     public void showDetails() {
4         System.out.println("This is a BMW vehicle.");
5     }
6
7     public void maxSpeed() {
8         System.out.println("Speed varies by model.");
9     }
10 }
11
12 // Subclasses overriding methods
13 class BMWSeries3 extends BMW {
14     @Override
15     public void showDetails() {
16         System.out.println("This is a BMW Series 3.");
17     }
18
19     @Override
20     public void maxSpeed() {
21         System.out.println("Max Speed: 240 km/h");
22     }
23 }
24
25 class BMWSeries5 extends BMW {
26     @Override
27     public void showDetails() {
28         System.out.println("This is a BMW Series 5.");
29     }
30
31     @Override
32     public void maxSpeed() {
33         System.out.println("Max Speed: 260 km/h");
34     }
35 }
36
37 class BMWSeries7 extends BMW {
38     @Override
39     public void showDetails() {
40         System.out.println("This is a BMW Series 7.");
41     }
42
43     @Override
44     public void maxSpeed() {
45         System.out.println("Max Speed: 300 km/h");
46     }
47 }
48
49 // Main class to test runtime polymorphism
50 public class BMWTest {
51     public static void main(String[] args) {
52         BMW[] cars = { new BMWSeries3(), new BMWSeries5(), new BMWSeries7() };
53
54         for (BMW car : cars) {
55             car.showDetails();
56             car.maxSpeed();
57             System.out.println();
58         }
59     }
60 }
```

This is a BMW Series 3.
Max Speed: 240 km/h

This is a BMW Series 5.
Max Speed: 260 km/h

This is a BMW Series 7.
Max Speed: 300 km/h

...Program finished with exit code 0
Press ENTER to exit console.