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:

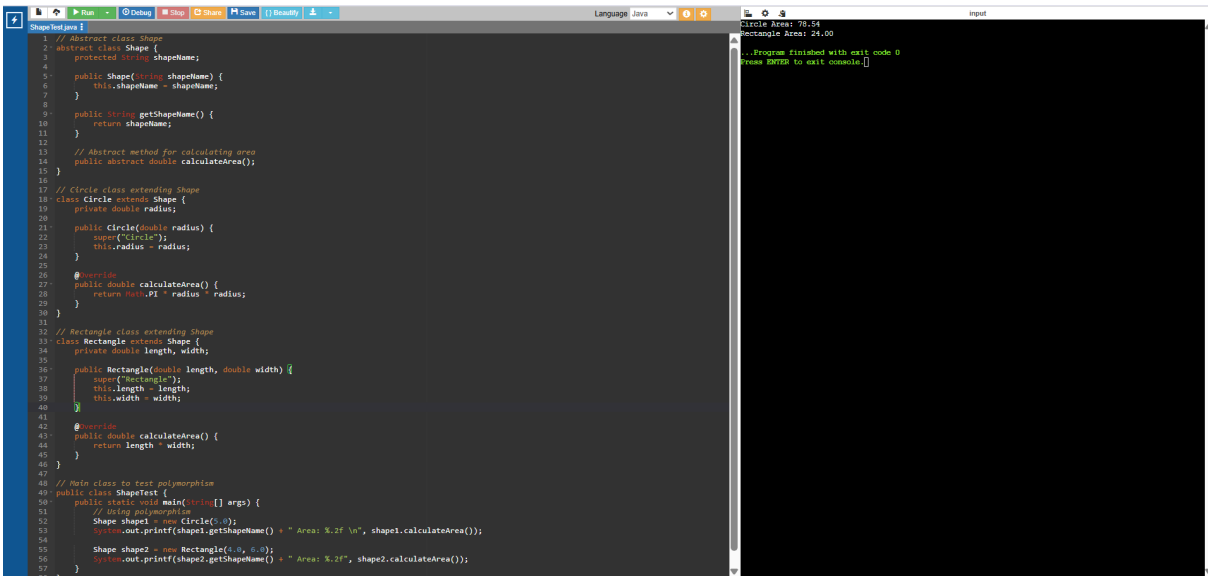
```
// Abstract class Shape
abstract class Shape {
    protected String shapeName;

    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



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

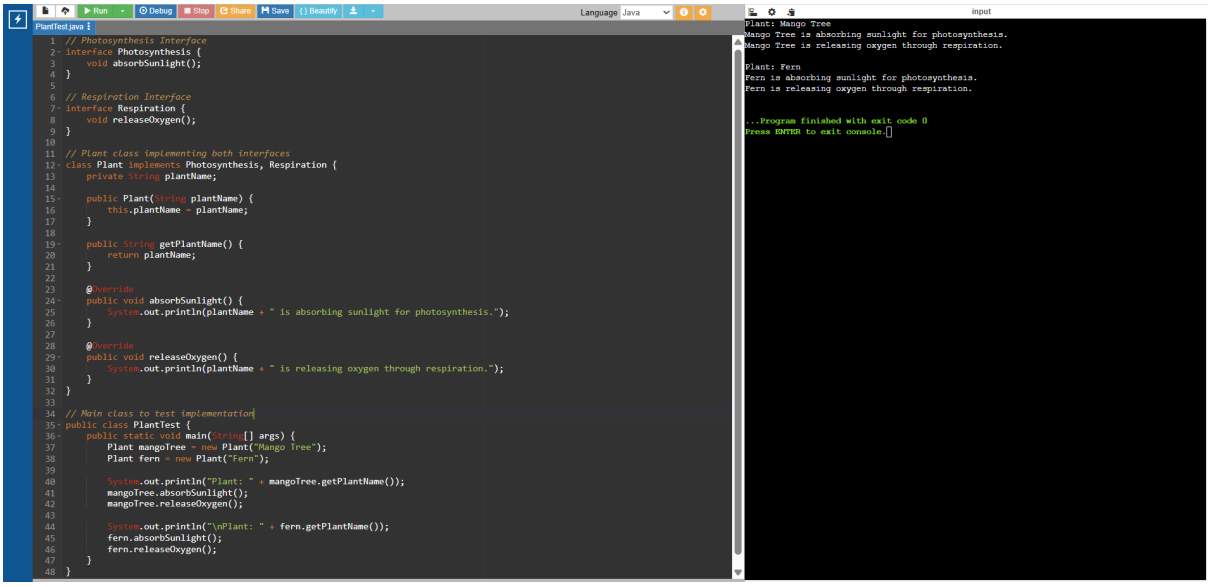
Sol:

```
// 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.");
    }
}
```

```java
// 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.
```



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