# Abstract class and Interface

# OBJECTIVES

- Understanding abstract methods and classes
- Declare and implement interface

## Program 1: Abstract Class

1. Create an abstract class called `Shape` with the following properties:
   - `name` (String): The name of the shape.
   - `color` (String): The color of the shape.
2. Declare an abstract method `calculateArea()` in the `Shape` class. This method should be responsible for calculating the area of the shape.
3. Create two concrete subclasses of `Shape`:
   - `Circle`: Include a `radius` property.
   - `Rectangle`: Include `length` and `width` properties.
4. Implement the `calculateArea()` method in both `Circle` and `Rectangle` classes to calculate the area of the respective shapes.

## Program 2: Interface

1. Create an interface called `Resizable` with the following method:
   - `resize(double factor)`: This method should resize the shape by the given factor.
2. Modify the `Rectangle` class to implement the `Resizable` interface. Implement the `resize(double factor)` method to adjust the length and width of the rectangle.

## Program 3: Testing

1. Create a `ShapeTest` class with the `main` method.
   - Create instances of `Circle` and `Rectangle`.
   - Set values for properties (name, color, radius, length, width).
   - Call the `calculateArea()` method for each shape and display the result.
2. Test the resizing functionality for a rectangle. Resize the rectangle by a factor, display the new length and width, and recalculate the area.

## Part 4: Reflection

1. In the `Shape` class, add a method called `displayShapeInfo()` that displays the name, color, and area of the shape. Invoke this method from the `ShapeTest` class to display information about each shape.
2. Reflect on the use of abstract classes and interfaces in the design. Discuss when it is appropriate to use an abstract class versus an interface in the context of this exercise. Note: Reflection is just another term for saying display function more technically, but it also means more. Refer *java.lang.reflect* package for more info. Sample usage Reflection in Java - GeeksforGeeksFor the scope of this exercise we will only deal with display function.

## Sample code

```
// Abstract class
abstract class Shape {
    protected String name;
    protected String color;

    public Shape(String name, String color) {
        this.name = name;
        this.color = color;
    }

    public abstract double calculateArea();

    // Additional method for displaying shape information
    public void displayShapeInfo() {
        System.out.println("Shape: " + name);
        System.out.println("Color: " + color);
        System.out.println("Area: " + calculateArea());
    }
}

// Concrete classes
class Circle extends Shape {
    private double radius;

    public Circle(String name, String color, double radius) {
        super(name, color);
        this.radius = radius;
    }

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

```
    }
}

class Rectangle extends Shape implements Resizable {
    private double length;
    private double width;

    public Rectangle(String name, String color, double length, double width) {
        super(name, color);
        this.length = length;
        this.width = width;
    }

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

    @Override
    public void resize(double factor) {
        this.length *= factor;
        this.width *= factor;
    }
}

// Interface
interface Resizable {
    void resize(double factor);
}

// Testing class
public class ShapeTest {
    public static void main(String[] args) {
        Circle circle = new Circle("Circle", "Red", 5.0);
        Rectangle rectangle = new Rectangle("Rectangle", "Blue", 4.0, 6.0);

        // Display information
        circle.displayShapeInfo();
        System.out.println("--------------------");
        rectangle.displayShapeInfo();

        // Resize rectangle and display updated information
        rectangle.resize(1.5);
        System.out.println("--------------------");
        rectangle.displayShapeInfo();
    }
```

```
}
```

# Homework

### Banking System Program Construction - Step-by-Step Instructions

In this guided exercise, you will construct a simplified banking system using Java, emphasizing the principles of abstract classes and interfaces. Follow these step-by-step instructions to build the program:

## Step 1: Create the `BankAccount` Abstract Class

1. Create a new Java class named `BankAccount`.
2. Inside the `BankAccount` class, declare the instance variables `accountNumber` (String) and `balance` (double).
3. Create a constructor to initialize these variables.
4. Implement a concrete method `displayAccountInfo()` that displays the account number and balance.
5. Add an abstract method `performAccountMaintenance()`. This method will represent account-specific maintenance operations and will be implemented by subclasses.

## Step 2: Implement the `Transaction` Interface

1. Create a new Java interface named `Transaction`.
2. Inside the `Transaction` interface, declare two abstract methods: `double deposit(double amount)` and `double withdraw(double amount)`.

## Step 3: Create the `SavingsAccount` Class

1. Create a new Java class named `SavingsAccount` that extends `BankAccount` and implements `Transaction`. i.e. `class SavingsAccount extends BankAccount implements Transaction`
2. Declare an additional instance variable, `interestRate` (double), specific to a savings account.
3. Implement the abstract method `performAccountMaintenance()` by printing a message specific to savings account maintenance.
4. Provide concrete implementations for the `deposit` and `withdraw` methods as per the requirements of the `Transaction` interface.
5. Introduce a method `applyInterest()` to demonstrate a unique operation for a savings account. Interest is earned so add the interest earned to the balance

### Step 4: Create a Test Class

1. Create a new Java class named `BankingSystem` for testing.
2. Inside the `BankingSystem` class, create an instance of `SavingsAccount`.
3. Perform operations such as deposits, withdrawals, and applying interest.
4. Display the account information at different stages using the `displayAccountInfo` method.
5. Call the `performAccountMaintenance` method to demonstrate the implementation of the abstract method.

### Step 5: Compile and Run

1. Compile all your Java classes.
2. Run the `BankingSystem` class.
3. Observe the output to ensure that the program functions as expected.

NOTE: Please include class diagram for all programs in Lab Report and add as much as description as you can.