Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester:(Spring, Year:2024), B.Sc. in CSE (Day)

Lab Report NO: 04

Course Title: Object-Oriented Programming Lab

Course Code: CSE 202 Section: 223 D9

Lab Experiment Name: Polymorphism.

Student Details

Name		ID
1.	Promod Chandra Das	231002005

Lab Date :

Submission Date :

Course Teacher's Name : Noyan Ali

Lab Report Status

Marks: …… **Signature:.....**

Comments:..... **Date:.....**

TITLE OF THE LAB REPORT EXPERIMENT:

- Polymorphism

OBJECTIVES/AIM :

- Understanding Polymorphism in Abstract classes
- Method overloading, Method overriding

PROCEDURE / ANALYSIS / DESIGN :

Problem analysis

Polymorphism allows us to use 1 interface with multiple implementations. Polymorphism are of 2 different
kinds:

- Runtime Polymorphism: This is also called dynamic polymorphism. The call to overridden methods are
resolved at runtime. Java achieves this by method overriding.
- Compiletime Polymorphism: It is also called static polymorphism. This is achieved by method overloading
or operator overloading. Java doesn't support the Operator Overloading.

Using both compliletime and runtime polymorphism to implement the following:

- Creating an abstract class Shape has to be created with one abstract method printshape()
- Creating subclass Circle of abstract class Shape
- Creating subclass Tetragon of abstract class Shape
- Override the printshape method in both Circle and Tetragon class
- Define method area() to calculate area of the tetragon. Use method overloading to calculate area of a
sqaure or rectangle based on input.

Abstract class

Java docs [1] states: An abstract class is a class that is declared abstract—it may or may not include abstract
methods. Abstract classes cannot be instantiated, but they can be subclassed.

An abstract method is a method that is declared without an implementation (without braces, and followed
by a semicolon), like this:

```
1 abstract void moveTo(double deltaX, double deltaY);
```

If a class includes abstract methods, then the class itself must be declared abstract, as in:

```
1 public abstract class GraphicObject {
2 // declare fields
3 // declare nonabstract methods
4 abstract void draw();
5 }
```

- **Subclass**

A class that inherits properties from another class are called as subclass. It is often the case that a subclass
inherits another subclass. If class B inherits from class A, then A is the superclass and B is the subclass.

```
1 /*java subclass declaration:*/
2 class MyClass extends MySuperClass {
3 // field, constructor, and
4 // method declarations
5 }
```

- **Constructor**

Constructors initializes an object when it is created. A constructors is a methods that has the same name as
the class itself. All the supeclass and subclass can have constructor.

```
1 /*java declaration:*/
2 class MyClass {
3 // constructor
4 Myclass(parameters){
5 //
6 }
7 }
```

- **Implementation**

• Create a java project name Lab(this also creates Lab.java file)
• Create abstract class Shape
• Create Circle subclass of Shape
• Create Tetragon subclass of Shape

• You have to override printshape() in both Circle and Tetragon
• Implement area() in both Circle and Tetragon
• Calculate area of either square or rectangle based on inputs (i.e Method overloading)

Lab.java file code:

```
1 /*Lab.java file*/
2
3
4 package lab;
5
6 public class lab {
7
8 public static void main(String[] args) {
9 Circle c=new Circle();
10 c.print_shape();
11 c.area(3.00);
12
13 Tetragon t=new Tetragon("Tetragon");
14 t.print_shape();
15 t.area(2);
16 t.area(2,3);
17 }
18
19 }
```

Create Shape.java

```
1
2 package lab;
3
4 abstract public class Shape {
5 abstract void print_shape();
6 }
```

❖ **Create Circle.java**

```
1
2 package lab;
3
4 public class Circle extends Shape{
5
6 final double PI=3.1415;
7
8 void print_shape(){
```

```
9 System.out.println("Circle");
10 }
11 void area(double r){
12 System.out.println("Circle area: "+(PI*Math.pow(r, 2)));
13 }
14 }
```

❖ **Create Tetragon.java**

```
1
2 package lab;
3
4 public class Tetragon extends Shape{
5
6 String s;
7
8 Tetragon(String s){
9 this.s=s;
10 }
11 void print_shape(){
12 System.out.println(this.s);
13 }
14
15 void area(double a){
16 System.out.println("Square: "+Math.pow(a, 2));
17 }
18 void area(double a, double b){
19 System.out.println("Rectangle: "+(a*b));
20 }
21
22 }
```

❖ **Input/Output**

Output of the program is given below.

```
abstract class
28.27431
Area of square: 4.0
Area of square: 6.0
```

❖ **Lab Exercise (Submit as a report)**

Suggested problem:

1. Write a java program that will create a class "Shape" from which you can create two objects like - "Rectangle"
and "Square". Add the following:

• Determine the attributes of both objects according to your choice (length and breadth).

• Initialize a parameterized constructors for both objects, that will receive the value of (length) and (length,
breadth) from main() function.

• Declare two overloading methods "CalculateArea()" and two overloading methods "CalculatePerimeter()"
in the Shape class. Return the calculated area and perimeter in main() function for both Square and
Rectangle, with the help of a object and print the values.

❖ **Code:**

```
// Define the Shape class
class Shape {
    // Define attributes
    protected double length;
    protected double breadth;

    // Parameterized constructor for Rectangle
    public Shape(double length, double breadth) {
        this.length = length;
        this.breadth = breadth;
    }

    // Parameterized constructor for Square
    public Shape(double length) {
        this.length = length;
```

```
        this.breadth = length;
    }

    // Overloaded method to calculate area for Rectangle
    public double calculateArea() {
        return length * breadth;
    }

    // Overloaded method to calculate area for Square
    public double calculateArea(double side) {
        return side * side;
    }

    // Overloaded method to calculate perimeter for Rectangle
    public double calculatePerimeter() {
        return 2 * (length + breadth);
    }

    // Overloaded method to calculate perimeter for Square
    public double calculatePerimeter(double side) {
        return 4 * side;
    }
}

public class Main {
    public static void main(String[] args) {
        // Create a Rectangle object
        Shape rectangle = new Shape(5.0, 3.0);
        // Calculate and print area and perimeter of the rectangle
        System.out.println("Rectangle Area: " + rectangle.calculateArea());
        System.out.println("Rectangle Perimeter: " + rectangle.calculatePerimeter());

        // Create a Square object
        Shape square = new Shape(4.0);
        // Calculate and print area and perimeter of the square
        System.out.println("Square Area: " + square.calculateArea(4.0));
        System.out.println("Square Perimeter: " + square.calculatePerimeter(4.0));
    }
}
```

❖ Discussion:

Polymorphism, meaning "many forms" in Greek, is a powerful concept in object-oriented programming (OOP) that allows objects of different classes to respond differently to the same method call. It fosters flexibility and code reusability.

- Discuss the difference between method overriding and overloading and provide real-world examples.
- Explore how polymorphism enables the "write once, run anywhere" principle in object-oriented programming.
- Analyze the role of interfaces in achieving polymorphism. How do interfaces contribute to loose coupling?
- Discuss the benefits and limitations of runtime vs. compile-time polymorphism.
- Provide code examples demonstrating different aspects of polymorphism (e.g., shape hierarchy with various `draw()` methods).