Department of
Computer Science and Engineering

Title: Polymorphism

Object Oriented Programming Lab
CSE 202

Green University of Bangladesh

1 Objective(s)

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

2 Problem Statement

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.

3 Terms

3.1 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 }
```

3.2 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 }
```

3.3 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.

```
/*java declaration:*/
class MyClass {
    //  constructor
    Myclass(parameters){
    //
    }
}
```

4 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:

```
/*Lab.java file*/


package lab;

public class lab {

    public static void main(String[] args) {
        Circle c=new Circle();
        c.print_shape();
        c.area(3.00);

        Tetragon t=new Tetragon("Tetragon");
        t.print_shape();
        t.area(2);
        t.area(2,3);
    }

}
```

Create Shape.java

```

package lab;

abstract public class Shape {
    abstract void print_shape();
}
```

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 }
```

5 Input/Output

Output of the program is given below.

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

6 Discussion

We have an abstract class Shape and abstract method shape(). Circle and Tetragon both used method overriding. This is also needed and shape() method was declared abstract. Method overloading is used in Tetragon to calculate area of square or rectangle based on input.

7 Lab Task (Please implement yourself and show the output to the instructor)

1. Specific problem/problems provided by the instructor.

7.1 Problem analysis

- Is it necessary to use abstract method in abstract class?
- Can abstract classes be subclass of other abstract class?
- Is it necessary to override abstract method in subclass?

8 Lab Exercise (Submit as a report)

Specific problem/problems for lab report may be provided by course teacher. Some suggested problems:

- Write code to demonstrate both method overriding and overloading.

9 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected. Any other policy applied by course teacher may hold applicable.

10 References

https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html

11 Appendix

If need be, course instructor might provide some additional materials.