Metropolitan State University, St. Paul, Minnesota
ICS 372 Object-Oriented Design and Implementation
Programming Assignment 3

Due: 11:59 PM on February 19, 2021
Maximum Points: 50

In this assignment, you will program using the Adapter pattern. The requirements and steps are given below.

1) First, copy the Point class from Assignment 1 to the new project. (Be sure to name the project correctly.)
2) Next, create an interface named Shape having just one method named display(), with no parameters and returning void. A Shape object represents a shape (like a line or circle or rectangle) that can be displayed graphically or by listing its relevant points on the console. The display mode (graphical or listing) is at the discretion of the implementor.
3) The interface is implemented by the following class Polygon, which represents a polygon. The Point class is from the first assignment. (Important convention: X coordinates increase as we move right and Y coordinates increase as we move **down**.)

```
/**
 * This class implements a Polygon shape. The vertices are stored in an array of Point objects.
 * Vertices are added using the add() method.
 * After all vertices are added, the user is expected to call the end() method, which duplicates the first vertex
 * and stores it as the last vertex. This makes it a bit easier to display the Polygon in a graphical interface.
 * (The current implementation only displays the vertices on the console.)
 * @author Brahma Dathan
 * @Copyright (c) 2018
*/

public class Polygon implements Shape {

   private Point[] points;
   private int numberOfPoints;
   /**
    * Creates the array to hold at most 10 Point objects. The addPoint() method
    * expands the size as needed.
    */
   public Polygon() {
        points = new Point[10];
   }

   /**
    * Adds one more vertex to the polygon
    *
    * @param point
    */
   public void addPoint(Point point) {
        if (this.points.length == numberOfPoints + 1) {
             Point[] temp = new Point[numberOfPoints * 2];
             System.arraycopy(points, 0, temp, 0, numberOfPoints);
             points = temp;
        }
        points[numberOfPoints++] = point;
        points[numberOfPoints] = point;
   }

   /**
```

```
     * Adds the first vertex as the last vertex, so the drawing (if we are drawing) would be complete.
     */
    public void end() {
        points[numberOfPoints] = points[0];
    }

    @Override
    public void display() {
        System.out.println("Polygon " + toString());
    }

    @Override
    public String toString() {
        String string = "Vertices ";
        for (int index = 0; index < numberOfPoints; index++) {
            string += "(" + points[index].x + ", " + points[index].y + ") ";
        }
        return string;
    }
}
```

4) Extend the Shape interface to create an interface named FilledShape to represent a shape that can be filled with a color the user chooses. This interface has the additional method named setColor() with 3 parameters: red, green, and blue, all of type int and can have values between 0 and 255 (both end values included) to represent the color of the shape. The implementing class is required to validate the values of the parameters before setting the color.
5) Create a class named Rectangle, which should be an **object** adapter to implement FilledShape and use Polygon as the adaptee.
   a) Have the necessary field(s) for the Adapter pattern.
   b) Have the necessary fields for storing the color.
   c) Create a constructor with following signature. The parameters point1 and point2 are opposite vertices of the rectangle. The constructor computes the other two vertices of the rectangle and uses the adaptee (using the constructor and appropriate methods) to store these vertices. (You must devise the logic to generate these Point objects.) The implementation should follow the expectations imposed by the adaptee.

      ```
      public Rectangle(Point point1, Point point2) throws Exception {
      ```

      If the two Point objects do not represent a rectangle with non-zero area (for example, point1 is the same as point2), the constructor throws an exception.
      The vertices **must** be stored in the clockwise direction with point1 as the first vertex and point2 as the third vertex. (Remember the convention regarding x and y coordinates in Step 3.)
   d) Override the display() method, so it works as given in the example.
   e) Override the toString() method, so it works as given in the example.
   f) No other fields or methods are allowed in the class.
6) Implement the FilledShape interface to create a class named Square to represent a square. This class should use the Rectangle class as a class adaptee. It must follow the requirements given below.
   a) Create a constructor with following signature. The parameters point1 and point2 are opposite vertices of the square. The constructor generates the other two vertices of the square and uses the adaptee to store these vertices. (You must devise the logic to generate these Point objects.)

`public Square(Point point1, Point point2) throws Exception {`

If the two Point objects do not represent a square with non-zero area (for example, point1 is the same as point2) or a true square (the sides are not equal), the constructor throws an exception. The vertices **must** be stored in the clockwise direction with point1 as the first vertex and point2 as the third vertex. (Again, remember the convention regarding x and y coordinates in Step 3).

b) Override the display() method, so it works as given in the example.
c) No other fields or methods are allowed in the class.

7) Test your implementation thoroughly. If you have any questions regarding the steps/directions, you must contact me in a timely manner.
8) Submit the Java Eclipse project.

Example

When the following code is executed,

```
public class DathanDriver {
    public static void main(String[] args) throws Exception {
        Polygon polygon = new Polygon();
        polygon.addPoint(new Point(3, 4));
        polygon.addPoint(new Point(9, 8));
        polygon.addPoint(new Point(2, 7));
        polygon.addPoint(new Point(15, 6));
        polygon.end();
        polygon.display();
        System.out.println("Polygon toString " + polygon.toString());
        Rectangle rectangle = new Rectangle(new Point(1, 2), new Point(5, 8));
        rectangle.display();
        System.out.println(rectangle);
        Rectangle rectangle2 = new Rectangle(new Point(5, 8), new Point(1, 2));
        rectangle2.display();
        System.out.println(rectangle2);
        Rectangle rectangle3 = new Rectangle(new Point(1, 8), new Point(5, 2));
        rectangle3.display();
        rectangle3.setColor(100, 200, 150);
        rectangle3.setColor(256, 100, 100);
        rectangle3.setColor(100, 256, 100);
        rectangle3.setColor(120, 100, 256);
        rectangle3.setColor(-1, 100, 100);
        rectangle3.setColor(100, -1, 100);
        rectangle3.setColor(120, 100, -1);
        System.out.println(rectangle3);
        Rectangle rectangle4 = new Rectangle(new Point(5, 2), new Point(1, 8));
        rectangle4.display();
        System.out.println(rectangle4);
        try {
            Rectangle rectangle5 = new Rectangle(new Point(8, 19), new Point(8, 30));
            rectangle5.display();
        } catch (Exception e) {
            System.out.println("throws exception for rectangle");
        }
        Square square1 = new Square(new Point(4, 5), new Point(8, 9));
        square1.display();
        System.out.println(square1);
        try {
            Square square2 = new Square(new Point(8, 19), new Point(10, 30));
            square2.display();
        } catch (Exception e) {
            System.out.println("throws exception for square");
        }
```

```
   }
}
```

The program should display the following.

```
Polygon Vertices (3, 4) (9, 8) (2, 7) (15, 6)
Polygon toString Vertices (3, 4) (9, 8) (2, 7) (15, 6)
Rectangle Vertices (1, 2) (5, 2) (5, 8) (1, 8)  filled in (r = 0, g = 0, b = 0)
Vertices (1, 2) (5, 2) (5, 8) (1, 8)  filled in (r = 0, g = 0, b = 0)
Rectangle Vertices (5, 8) (1, 8) (1, 2) (5, 2)  filled in (r = 0, g = 0, b = 0)
Vertices (5, 8) (1, 8) (1, 2) (5, 2)  filled in (r = 0, g = 0, b = 0)
Rectangle Vertices (1, 8) (1, 2) (5, 2) (5, 8)  filled in (r = 0, g = 0, b = 0)
Vertices (1, 8) (1, 2) (5, 2) (5, 8)  filled in (r = 100, g = 200, b = 150)
Rectangle Vertices (5, 2) (5, 8) (1, 8) (1, 2)  filled in (r = 0, g = 0, b = 0)
Vertices (5, 2) (5, 8) (1, 8) (1, 2)  filled in (r = 0, g = 0, b = 0)
throws exception for rectangle
Square Vertices (4, 5) (8, 5) (8, 9) (4, 9)  filled in (r = 0, g = 0, b = 0)
Vertices (4, 5) (8, 5) (8, 9) (4, 9)  filled in (r = 0, g = 0, b = 0)
throws exception for square
```

The above example does not necessarily exercise the functionality thoroughly, including the situations that raise exceptions when the parameters to the Rectangle/Square constructors are inappropriate, color change, etc.

Submission

Submit the **Eclipse Java project**.

Follow the following steps to upload your code to D2L:

- Create a java project and call it `<yourLastName>Assignment3` (e.g., mine will be called `DathanAssignment3`). If you wish, you can add a qualifier to add the course name to have the form `<yourLastName>372Assignment3.`
- Create/add the .java files to implement the classes as described above.
- Archive the project into **one zip** file using Eclipse using the following steps:
    - In Eclipse Project Explorer, right click on the **project folder** of the project and click on Export.
    - Choose General then Archive File and click Next.
    - Use the Browse key to choose a folder to store the archive file on your hard drive and give the file the same name as your project (e.g., `DathanAssginment3.zip`), then click Save, then click Finish. Upload the **.zip** file you created to the D2L folder called `Assignment 3.`
    - The most recent submission will be graded, unless you indicate so in the submission page.

Grading

The project must be named as specified above. If you don't follow this naming convention or don't submit an Eclipse project, there will be a penalty of 10%.

Interfaces created as required	4
Rectangle class implemented as required	16
Square class implemented as required	10
Correctness	10
Coding Conventions and Documentation	10