Programmation 2 : Corrigé du TD 4
Code UE : SIN3U02

Interfaces

1 Interface `StringTransform`

1. Interface `StringTransform`

```
public interface StringTransform {
    public String applyTo(String s);
}
```

2. Classes implémentant `StringTransform` :

```
public class UpperCaseStringTransform implements StringTransform{
    @Override
    public String applyTo(String s) {
        return s.toUpperCase();
    }
}

public class LowerCaseStringTransform implements StringTransform {
    @Override
    public String applyTo(String s) {
        return s.toLowerCase();
    }
}

public class PrefixStringTransform implements StringTransform {
    int n;

    public PrefixStringTransform(int n) {
        this.n = n;
    }

    public String applyTo(String s) {
        return s.substring(0, n);
    }
}

public class PostfixStringTransform implements StringTransform{
    int n;

    @Override
    public String applyTo(String s) {
        return s.substring(s.length()-n);
    }

    public PostfixStringTransform(int n) {
        this.n = n;
    }
}
```

3. méthode statique `String[] applyTransformToStrings(String[] strings, StringTransform transform)`

```
public static String[] applyTransformToStrings(String[] strings,
                                                StringTransform transform){
    String[] transformedStrings = new String[strings.length];
    for(int index = 0; index < transformedStrings.length; index++){
        transformedStrings[index] = transform.applyTo(strings[index]);
    }
    return transformedStrings;
}
```

4. Classe `CompositeStringTransform`

```
public class CompositeStringTransform implements StringTransform{
    private StringTransform[] transforms;

    public CompositeStringTransform(StringTransform[] transforms){
        this.transforms = transforms.clone();
    }

    @Override
    public String applyTo(String s) {
        String result = s;
        for(int index =0; index<transforms.length; index++){
            result = transforms[index].applyTo(result);
        }
        return result;
    }
}
```

2 Interface Shape

2.1 Triangle

```
import tp2.lib.Painter;

import java.awt.*;

public class Triangle implements Shape {
    private final Point[] vertices;

    public Triangle(Point p1, Point p2, Point p3) {
        vertices = new Point[]{p1, p2, p3};
    }

    private Triangle(Point[] vertices){
        this.vertices = vertices;
    }

    @Override
```

```
    public double getPerimeter() {
        double sum = 0;
        for(int i=0; i<vertices.length; i++){
            sum += vertices[i].distanceTo(
                    vertices[(i+1)%vertices.length]);
        }
        return sum;
    }

    @Override
    public void draw(Painter painter, Color color) {
        for(int i=0; i<vertices.length; i++){
            vertices[i].drawLine(vertices[(i+1)%vertices.length],
                    painter, color);
        }
    }

    @Override
    public Shape translate(int dx, int dy) {
        Point[] vertices = new Point[3];
        for(int i=0; i<vertices.length; i++){
            vertices[i] = this.vertices[i].translate(dx, dy);
        }
        return new Triangle(vertices);
    }

    @Override
    public double getArea() {
        double p = getPerimeter()/2;
        double result = p;
        for(int i=0; i<vertices.length; i++){
            result *= p - vertices[i].distanceTo(
                    vertices[(i+1)%vertices.length]);
        }
        return Math.sqrt(result);
    }
}
```

2.2 Rectangle

```
import tp2.lib.Painter;

import java.awt.*;

public class Rectangle implements Shape{
    Point p1, p2;
    public Rectangle(Point p1, Point p2){
        this.p1 = p1;
        this.p2 = p2;
    }
```

```
    private int height(){
        return Math.abs(p1.y-p2.y);
    }

    private int width(){
        return Math.abs(p1.x-p2.x);
    }

    @Override
    public double getPerimeter() {
        return 2*width() + 2*height();
    }

    @Override
    public void draw(Painter painter, Color color) {
        int maxX = Math.max(p1.x, p2.x);
        int maxY = Math.max(p1.y, p2.y);
        int minX = Math.min(p1.x, p2.x);
        int minY = Math.min(p1.y, p2.y);
        Point [] vertices = new Point[]{
                new Point(maxX, maxY),
                new Point(maxX, minY),
                new Point(minX, minY),
                new Point(minX, maxY)
        };
        for(int i=0; i<vertices.length; i++){
            vertices[i].drawLine(vertices[(i+1)%vertices.length],
                    painter, color);
        }
    }

    @Override
    public Shape translate(int dx, int dy) {
        return new Rectangle(p1.translate(dx, dy),
                p2.translate(dx, dy));
    }

    @Override
    public double getArea() {
        return width() * height();
    }
}
```