Lab exercise 4: Implementing a design based on inheritance
24292-Object Oriented Programming

Jordi Polo Tormo : 239980
Simón Gasión Medina: 240126

1.Introduction

Este lab consiste en implementar una jerarquía de clases relacionadas entre sí por herencia, el objetivo es conseguir una interfaz que muestre una série de entidades en un plano de dos dimensiones y permitir al usuario interactuar con ellas moviéndolas una distancia a elegir mediante la interfaz que ya viene implementada al programa. Se pueden mover las entidades especificas que seleccione el usuario, ya que hemos implementado la parte opcional del lab.

“Entity” és una forma genérica de referirse a los diferentes elementos dibujados, que pueden ser un “PolygonalRegion” (igual que en labs anteriores), uno de dos subtipos de región polygonal denominados “TriangularRegion” y “RectangularRegion”, todos ellos formados a partir de una lista de puntos, o también puede ser una “EllipsoidalRegion” o “CircularRegion”, que a diferencia de las regiones poligonales se dibujan a partir de un solo punto que indica el centro de la ellipsoide y dos valores que deciden los radios del eje horizontal y vertical (un mismo valor para ambos radios en el caso de la region circular).

Cada Entidad también tiene dos valores de la clase “Color” para decidir su color de relleno y el color del delineado, así como la capacidad de calcular su propia área.

2.Description

Point y Vector:
La clase “Point” recibe dos nuevos métodos para ésta práctica que no tenía en las anteriores. El primero, “move”, aplica un cambio de coordenadas positivo o negativo en cada eje sobre el mismo punto, mientras que el vector difference toma otro punto como argumento y devuelve el camino de uno a otro en forma de “Vector”, una clase que expresa un movimiento. Ésta clase contiene un solo método, “Product”, que toma otro vector y devuelve el producto entre ambos.

```java
public class Point {
    //Atributes//
    protected int x;
    protected int y;
    //Methods//
    public Point(int xi, int yi) {
        this.x = xi;
        this.y = yi;
    }
    public int getX() {
        return this.x;
    }
    public int getY() {
        return this.y;
    }
    public void move(int xsum, int ysum){
        this.x = this.x + xsum;
        this.y = this.y + ysum;
    }
    public Vector difference(Point p) {
        Point q  = new Point(this.x, this.y);
        Vector v = new Vector(p.getX()-q.getX(), p.getY()-q.getY());
        return v;
    }
}
```

Entity y Region

“Entity” es una clase abstracta que actúa como base en la jerarquía de herencias, recibe al constructor el color del delineado y tiene getter y setter para interactuar con este atributo. Los demás métodos de la clase són abstractos, por lo que aún no tienen funcionalidad definida, pero un cambio que hemos hecho en esta clase respecto al esquema aportado en el enunciado del lab és que el método abstracto “isPointInside” se declara en “Entity” en vez de en “Region”, ésto es debido a que para implementar la función “mousePressed” de la parte opcional del lab necesitamos aplicar isPointInside a entidades, no a regiones, entonces al no existir el método para la clase entidad no encontramos otra manera de resolver el problema.

“Region” és otra clase abstracta que inherita “Entity”, por lo que también necesita un color para el constructor, pero además del color de delineado aquí se define también el color de relleno de la región. Si no se define mediante el método “setFillColor” un color para el atributo “FillColor”, automáticamente se le asignará el mismo color que el de delineado para evitar problemas a la hora de dibujar varias regiones, ya que de la manera que se implementan las diferentes instancias del método draw si el color de relleno tiene un valor nulo el programa usará el último color para rellenar, és decir, el color de delineado de la última entidad imprimida.

```java
import java.awt.*;

abstract public class Entity {
    protected Color lineColor;
    private double alpha;

    public Entity( Color lcinit ) {
        lineColor = lcinit;
    }

    abstract public void draw( java.awt.Graphics g );

    public void SetColor(Color c) {
        this.lineColor = c;
    }
    public Color GetColor() {
        return this.lineColor;
    }

    abstract public void move(Vector v);

    abstract public Boolean isSelected(Point p);

    public abstract Boolean isPointInside(Point p); //Pa

}
```

```java
import java.awt.Color;
abstract public class Region extends Entity {
    protected Color fillColor;
    public Region( Color lcinit ) {
        super(lcinit);
        this.fillColor = lcinit; //si se desea u
    }
    public abstract  double getArea();
    public void setFillColor(Color fillColor) {
        this.fillColor = fillColor;
    }
    //public abstract Boolean isPointInside(Poin
}
```

PolygonalRegion y ElipsoidalRegion;

La implementación de “PolygonalRegion” es muy similar a como era en el lab 3, la mayor diferencia es cómo “draw” ahora utiliza el método de “drawPolygon” en vez de “drawLine” como en otros labs. Ésto se debe a que para dibujar el color de relleno por debajo de la figura se necesitan dos vectores que contengan las coordenadas de X e Y respectivamente en vez de una série de puntos, que són los mismos requisitos de la función drawPolygon por lo que hemos adaptado el método entero. El método “getArea” és igual que en labs anteriores porque seguimos asumiendo sólo polígonos convexos.

En ambas clases “PolygonalRegion” y “EllipsoidalRegion” se definen los métodos abstractos declarados anteriormente, éstos són:

- “Move”, que aplica la función del mismo nombre a cada punto de la figura o al singular punto central de la elipse

- “IsPointInside”, que comprueba una série de condiciones diferentes según el tipo de la región proporcionadas en el enunciado del lab para comprobar si un determinado punto está dentro de la región en las coordenadas cartesianas.
- “IsSelected”, que no hemos visto la necesidad de implementar.

```java
public class PolygonalRegion extends Region {
    //Atributos//
    protected LinkedList<Point> points;
    //Métodos//
    public PolygonalRegion(LinkedList<Point> list, Color l) {
        super(l);
        this.points = list;
        if (points.size() > 0) {
            this.points.add(list.get(0)); //para calcular el area y dibujar es

        }
    }
    public double getArea() { //traducción a código de la formula del área de u
        double area = 0;
        for (int i = 0; i < this.points.size()-1; i++) {
            area += this.points.get(i).getX() * this.points.get(i+1).getY();
        }
        for (int i = 0; i < this.points.size()-1; i++) {
            area -= this.points.get(i).getY() * this.points.get(i+1).getX();
        }
        area = 0.5 * area;
        if(area < -1){
            area = area * (-1);
            return area;
        }
        else{
            return area;
        }
    }
```

```java
public void draw(java.awt.Graphics g) {
    int size = this.points.size();
    int[] X;
    int[] Y;
    X = new int[size];
    Y = new int[size];
    for (int i = 0; i < (size); i++) {
        X[i] = this.points.get(i).getX();
        Y[i] = this.points.get(i).getY();
    }
    g.setColor(this.fillColor);
    g.fillPolygon(X, Y, size);
    g.setColor(this.lineColor);
    g.drawPolygon(X, Y, size);
}
@Override
public void move(Vector v)  {
    int size = this.points.size();
    for (int i = 0; i < (size-1); i++) {
        points.get(i).move(v.x, v.y);
    }
}
```

```java
@Override
public Boolean isPointInside(Point p) {
    int size = this.points.size();
    double prevSign = 0;
    Boolean check = true;
    int product = 0;
    Point q1;
    Point q2;
    Vector v1;
    Vector v2;
    for (int i = 0; i < (size-1); i++) {
        q1 = points.get(i);
        q2 = points.get(i+1);
        v1 = q1.difference(q2);
        v2 = q1.difference(p);
        product = v1.Product(v2);
        if (prevSign == 0) {
            prevSign = Math.signum(product);
        }
        else {
            check = (Math.signum(product) == prevSign);
            prevSign = Math.signum(product);
        }
        if (check == false) {
            return check;
        }
    }
    return check;
}
```

```java
public class EllipsoidalRegion extends Region{
    private Point c;
    private int r1;
    private int r2;
    public EllipsoidalRegion(Point c, int r1, int r2, Color lc) {
        super(lc);
        this.c = c;
        this.r1 = r1;
        this.r2 = r2;
    }
    public double getArea() {
        double Area =  Math.PI*r1*r2;
        return Area;
    }
    public void draw(java.awt.Graphics g) {
        g.setColor(this.fillColor);
        g.fillOval(this.c.getX(), this.c.getY(), r1, r2);
        g.setColor(this.lineColor);
        g.drawOval(this.c.getX(), this.c.getY(), r1, r2);
    }
    @Override
    public void move(Vector v)  {
        this.c.move(v.x, v.y);
    }
    @Override
    public Boolean isPointInside(Point p) {
        Boolean check;
        check = ((Math.pow(p.getX() - c.getX(), 2)/Math.pow(r1, 2)) + (Math.pow(p.getY() - c.getY(), 2)/Math.pow(r2, 2)) <= 1);
        return check;
    }
```

TriangularRegion, RectangularRegion y CircularRegion:

Al fin y al cabo estas tres clases son un subset específico de “PolygonalRegion” o “EllipsoidalRegion”, y cualquier elemento que se cree aquí se puede también crear en sus clases super fácilmente. “RectangularRegion” ha sido con diferencia la más complicada de implementar puesto que solo se lo pasan dos puntos, las esquinas del rectangulo, al constructor, por lo que en el constructor hay que utilizar las coordenadas X e Y de los dos puntos que se proporcionan al constructor para crear dos nuevos puntos y completar el rectángulo.

```java
public class TriangularRegion extends PolygonalRegion {
    //Atributos//
    private LinkedList<Point> points;
    //Métodos//
    public TriangularRegion(Point p1, Point p2, Point p3, Color lc) {
        super(new LinkedList<Point>(), lc);
        super.points.add(p1);
        super.points.add(p2);
        super.points.add(p3);
        super.points.add(p1); //para calcular el area y dibujar es impr
    }
    public double getArea() { //traducción a código de la formula del á
        return super.getArea();
    }
}
```

```java
public class RectangularRegion extends PolygonalRegion {
    //Atributos//
    //Métodos//
    public RectangularRegion(Point p1, Point p3, Color lc) {
        super(new LinkedList<Point>(), lc);
        this.points.add(p1);
        this.points.add(new Point(p1.getX(), p3.getY()));
        this.points.add(p3);
        this.points.add(new Point(p3.getX(), p1.getY()));
        this.points.add(p1);
    }
    public double getArea() { //traducción a código de la formul
        return super.getArea();
    }
}
```

```java
public class CircularRegion extends EllipsoidalRegion {
    //Atributos//
    private Point center;
    private int radius;
    //Métodos//
    public CircularRegion(Point c, int r, Color lc) {
        super(c, r, r, lc);
    }
    public double getArea() { //traducción a código de la
        double Area =  Math.PI*Math.pow(this.radius, 2);
        return Area;
    }
}
```

GraphicalInterface:

La clase “GraphicalInterface” actúa como cuerpo principal del programa, para comprobar que todas las funciones y clases funcionan correctamente hemos creado una instancia de cada clase con la excepción de regiones poligonales y elipsoidales, de las cuales hemos creado dos. A cada region creada se le asigna uno o dos colores y se añaden a la lista de drawables.

```
EllipsoidalRegion circle = new CircularRegion(new Point(300, 230), 100, Color.BLUE);

LinkedList<Point>p = new LinkedList<Point>();
p.add(new Point(100, 150));
p.add(new Point(200, 230));
p.add(new Point(100, 280));
p.add(new Point(40, 200));
PolygonalRegion polygon = new PolygonalRegion(p, Color.GREEN);
polygon.setFillColor(Color.YELLOW);

LinkedList<Point>p2 = new LinkedList<Point>();
p2.add(new Point(200, 120));
p2.add(new Point(200, 230));
p2.add(new Point(100, 280));
p2.add(new Point(80, 230));
PolygonalRegion polygon2 = new PolygonalRegion(p2, Color.YELLOW);
polygon2.setFillColor(Color.GREEN);

RectangularRegion rectangle = new RectangularRegion(new Point(400, 400), new Point(300, 450), Color.black);
rectangle.setFillColor(Color.lightGray);

TriangularRegion triangle = new TriangularRegion(new Point(0, 70), new Point(0, 0), new Point(80, 50), Color.CYAN);
triangle.setFillColor(Color.black);

d.addDrawable(ellipse1);
d.addDrawable(ellipse2);
d.addDrawable(circle);
d.addDrawable(polygon);
d.addDrawable(polygon2);
d.addDrawable(rectangle);
d.addDrawable(triangle);
```

Parte opcional:

La parte opcional de éste lab se basaba en implementar la función de selección al programa, lo cual depende sobre todo de la función “mousePressed” de la librería MouseListener. Para implementarla simplemente se comprueba por cada entidad si el punto sobre el que ha clicado el ratón está dentro suyo, en cual caso se añade a la lista de entidades seleccionadas sobre las cuales se aplica cualquier cambio de posición que se indique en la interfaz del programa. No hemos implementado ninguna manera de des-seleccionar una entidad una vez ha sido seleccionada, ni de vaciar la lista ”selection”, puesto que no se pedía en el enunciado.

```
@Override
public void mousePressed(MouseEvent e) {
    Point mouse = new Point(e.getX(), e.getY());
    int size = this.drawables.size();
    for (int i = 0; i < (size); i++) {
        if (this.drawables.get(i).isPointInside(mouse) == true) {
            this.selection.addLast(this.drawables.get(i));
        }
    }
}
```

3.Conclusión

La práctica ha estado realizada de la manera correcta, no hemos encontrado ningún error notable y el programa se ejecuta y es completamente funcional sin ningún error.

No hemos encontrado ninguna dificultad que nos haya impedido seguir trabajando, pero al ser con diferencia el lab más laborioso hasta ahora y tener poca experiencia con las relaciones de herencia y clases abstractas en java han surgido algunos pequeños contratiempos al implementar los constructores de algunas clases (sobretodo el de RectangularRegion). También tuvimos que improvisar a la hora de implementar la librería MouseListener, puesto que siguiendo las instrucciones del apartado 5 del lab el entorno de programación mostraba errores en el importe de librerías.