Pablo Fernández
Clara López
S101, P101

Lab 4 Report

Introduction

This report presents the work done in the fourth laboratory project of the Object-Oriented Programming course.

The aim of this fourth project was to create a program that moves and draws the different types of regions. To do so, we had to implement the classes and relations we had designed in the previous seminar sessions.

In this new project we introduced a new concept, *abstract*. An abstract class is a restricted class that cannot be used to create objects. To access it, the class has to be inherited from another class. In this lab, we have two abstract classes, Entity, because its purpose is to represent objects that can be drawn on the screen or moved, and Region, because its purpose is to represent the different types of regions we have defined.

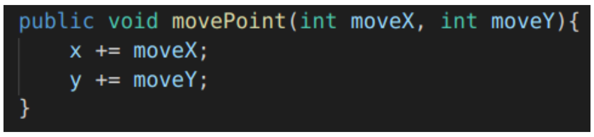
When creating a class, it is important to take into account the relation they may have between them. In this case, we can see that *Text, Line* and *Region* inherit from *Entity*, *PolygonalRegion* and *EllipsoidalRegion* inherit from *Region*, *RectangularRegion* and *TriangularRegion* inherit from *PolygonalRegion* and *CircularRegion* inherit from *EllipsoidalRegion*.
We can also see that between *Entity* and *DrawApp* there are two Aggregation relations of many to one, and also between *Color* and *Region* we can find an Association relation of one to one.
Moreover, *DrawApp* and *Entity* have a Dependency relation with *Graphics*, because they need this class in order to use the *draw* function.

Pablo Fernández
Clara López
S101, P101

Description

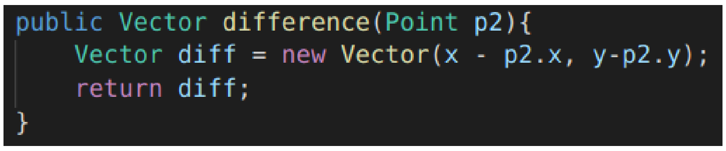
1. Points

In order to define the *Point* class we have used the functions defined in the previous projects, but we have added two new functions:



```java
public void movePoint(int moveX, int moveY){
    x += moveX;
    y += moveY;
}
```

The function movePoint is used to change the coordinates of the point in order to change its location.

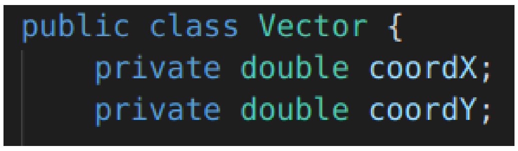


```java
public Vector difference(Point p2){
    Vector diff = new Vector(x - p2.x, y-p2.y);
    return diff;
}
```

The function difference is used to compute the difference between two points and create a vector with that coordinates.

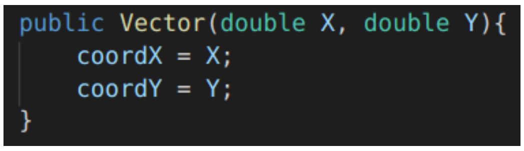
2. Vectors

We have created this class because it is easier to know whether a point lies inside a convex polygon by computing the cross product of vectors.



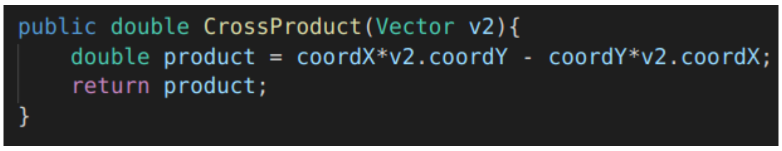
```java
public class Vector {
    private double coordX;
    private double coordY;
```

Here we have defined the attributes of this class, which are two variables that represents the coordinates of a vector



```java
public Vector(double X, double Y){
    coordX = X;
    coordY = Y;
}
```

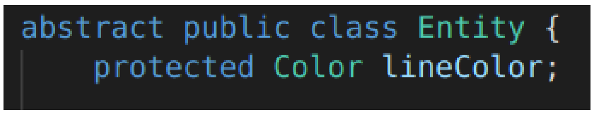
Constructor method, in order to assign values to the attributes.



```java
public double CrossProduct(Vector v2){
    double product = coordX*v2.coordY - coordY*v2.coordX;
    return product;
}
```

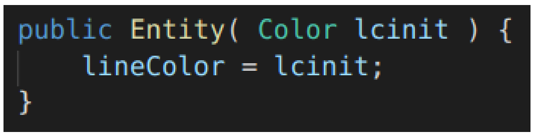
The function CrossProduct computes the cross product between two vectors.

3. Entity



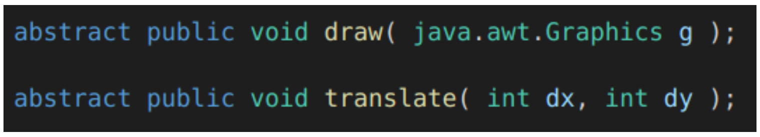
```java
abstract public class Entity {
    protected Color lineColor;
```

We have created the class *Entity* as an abstract class with the color of the line as an attribute. This attribute is *protected* since all the entities will have an associated line color.



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

Constructor method.

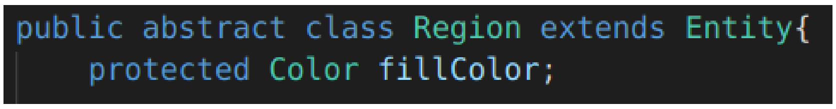


```
abstract public void draw( java.awt.Graphics g );

abstract public void translate( int dx, int dy );
```

These two functions, draw and translate are defined as abstract because all of the subclasses of Entity will perform these functions.

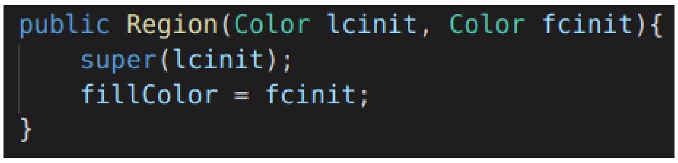
4. Region

In order to define this class we have used what we did in the previous projects but we have defined the class as an abstract class and the methods it had to abstract ones too.



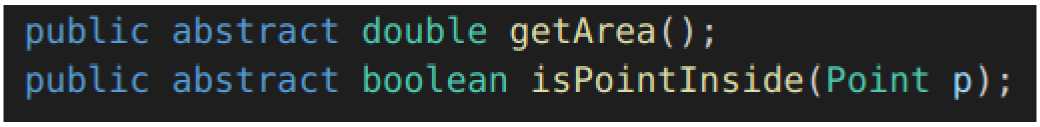
```
public abstract class Region extends Entity{
    protected Color fillColor;
```

Respect from the previous labs, we have added an attribute to the *Region* class. This attribute consists of the color to fill the different types of regions we have. Is defined as protected, because of the same reason that we have defined the attribute of entity as protected, because all of its subclasses have an associated fill color.



```
public Region(Color lcinit, Color fcinit){
    super(lcinit);
    fillColor = fcinit;
}
```

Constructor method. We use the command super in order to obtain the attribute of the *Entity* class from which *Region* inherits from.

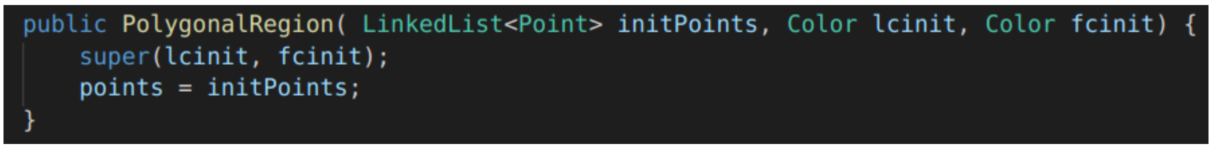


```
public abstract double getArea();
public abstract boolean isPointInside(Point p);
```

These two functions, getArea and isPointInside are defined as abstract because all of the classes that inherit from *Region* will perform these functions.

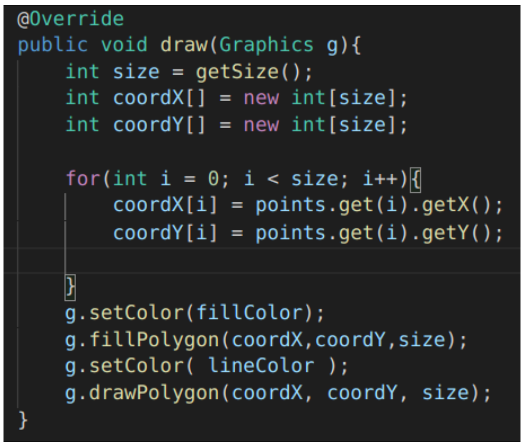
5. Polygonal Region

In order to define the *PolygonalRegion* class we have used the functions defined in the previous projects, but we have added new functions:



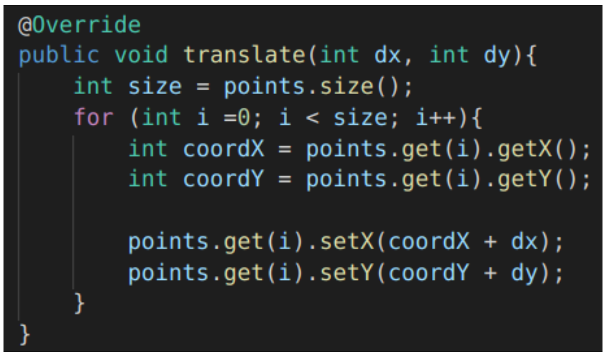
```
public PolygonalRegion( LinkedList<Point> initPoints, Color lcinit, Color fcinit) {
    super(lcinit, fcinit);
    points = initPoints;
}
```

First, we have changed the definition of the class, since now it has to get the attributes of *Region* and *Entity*, since this class inherits from *Region* which at the same time inherits from *Entity.*



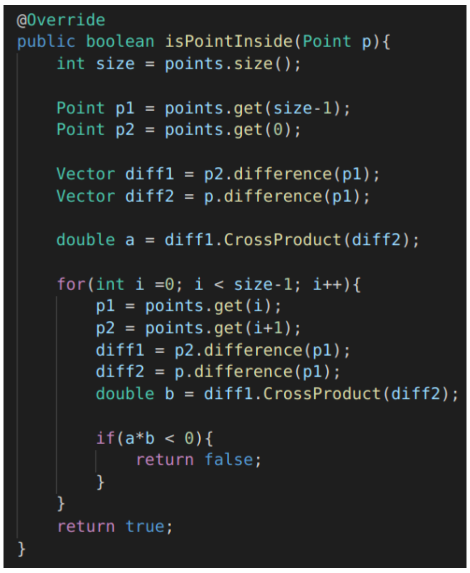
```java
@Override
public void draw(Graphics g){
    int size = getSize();
    int coordX[] = new int[size];
    int coordY[] = new int[size];

    for(int i = 0; i < size; i++){
        coordX[i] = points.get(i).getX();
        coordY[i] = points.get(i).getY();

    }
    g.setColor(fillColor);
    g.fillPolygon(coordX,coordY,size);
    g.setColor( lineColor );
    g.drawPolygon(coordX, coordY, size);
}
```

The function draw has the command *@Override* is to prevent that when we call this specific function it does not get the function defined in the class this class inherits from, in this case, in order not to get the draw function defined in *Region*.



```java
@Override
public void translate(int dx, int dy){
    int size = points.size();
    for (int i =0; i < size; i++){
        int coordX = points.get(i).getX();
        int coordY = points.get(i).getY();

        points.get(i).setX(coordX + dx);
        points.get(i).setY(coordY + dy);
    }
}
```

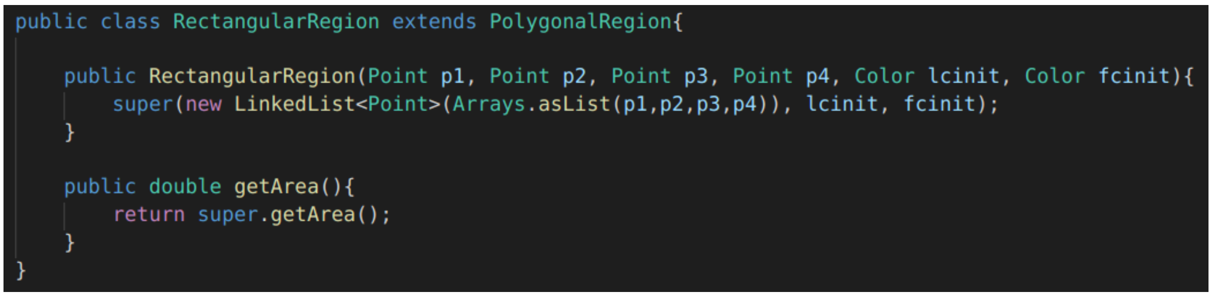
The function translate is used to move the region given a point along the x and y axis.



```java
@Override
public boolean isPointInside(Point p){
    int size = points.size();

    Point p1 = points.get(size-1);
    Point p2 = points.get(0);

    Vector diff1 = p2.difference(p1);
    Vector diff2 = p.difference(p1);

    double a = diff1.CrossProduct(diff2);

    for(int i =0; i < size-1; i++){
        p1 = points.get(i);
        p2 = points.get(i+1);
        diff1 = p2.difference(p1);
        diff2 = p.difference(p1);
        double b = diff1.CrossProduct(diff2);

        if(a*b < 0){
            return false;
        }
    }
    return true;
}
```

The IsPointInside function is used to know whether a point we are asking for belongs to the region. This is done by applying the instructions from the exercise. We convert the sides of the polygon to vectors, and we loop through them while creating a new vector that joins the point we aim to check and one of the points that form the polygon. Lastly, we check the result of cross product to see whether the point is inside or not.

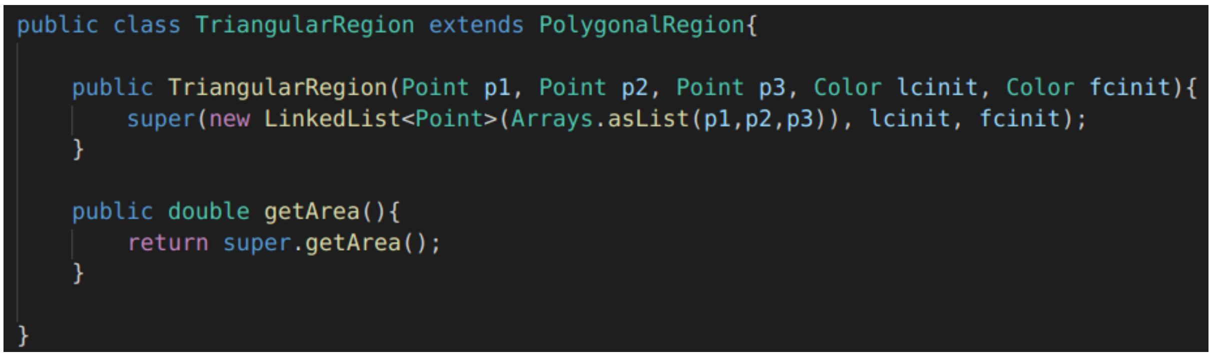
6. Rectangular Region



```
public class RectangularRegion extends PolygonalRegion{

    public RectangularRegion(Point p1, Point p2, Point p3, Point p4, Color lcinit, Color fcinit){
        super(new LinkedList<Point>(Arrays.asList(p1,p2,p3,p4)), lcinit, fcinit);
    }

    public double getArea(){
        return super.getArea();
    }
}
```

This class inherits from *PolygonalRegion*, which at the same time inherits from *Region* which at the same time inherits from *Entity*. That's why in it's construct method we can find the attributes defined in these three classes.
The function getArea uses the function defined in the *PolygonalRegion* class.

7. Triangular Region

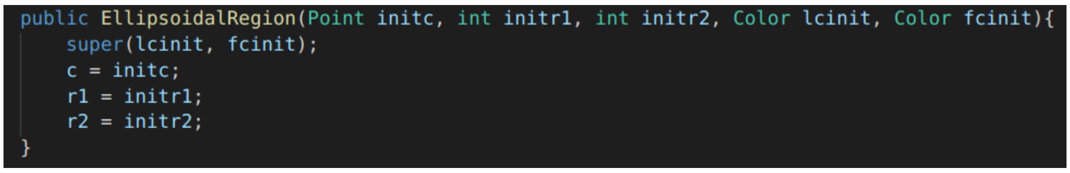


```
public class TriangularRegion extends PolygonalRegion{

    public TriangularRegion(Point p1, Point p2, Point p3, Color lcinit, Color fcinit){
        super(new LinkedList<Point>(Arrays.asList(p1,p2,p3)), lcinit, fcinit);
    }

    public double getArea(){
        return super.getArea();
    }

}
```

This class, the same way as the *RectangularRegion*, inherits from *PolygonalRegion*, which at the same time inherits from *Region* which at the same time inherits from *Entity*. That's why in it's construct method we can find the attributes defined in these three classes.
And the same way as the previous class commented, the function getArea uses the function defined in the *PolygonalRegion* class.

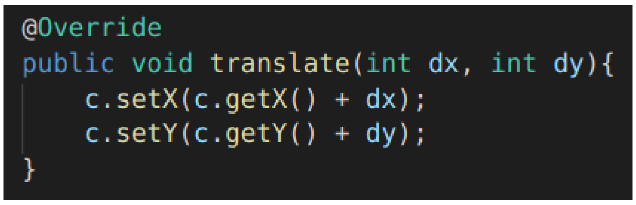
8. Ellipsoidal Region

In order to define the *EllipsoidalRegion* class we have used the functions defined in the previous projects, but we have added to new functions:



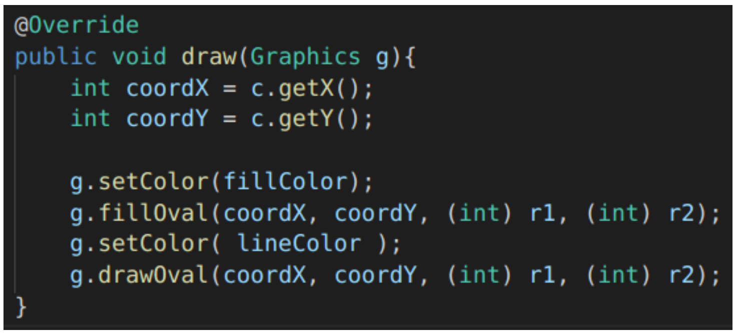
```
public EllipsoidalRegion(Point initc, int initr1, int initr2, Color lcinit, Color fcinit){
    super(lcinit, fcinit);
    c = initc;
    r1 = initr1;
    r2 = initr2;
}
```

First, we have changed the definition of the class, since now it has to get the attributes of *Region* and *Entity*, since this class inherits from *Region* which at the same time inherits from *Entity.*



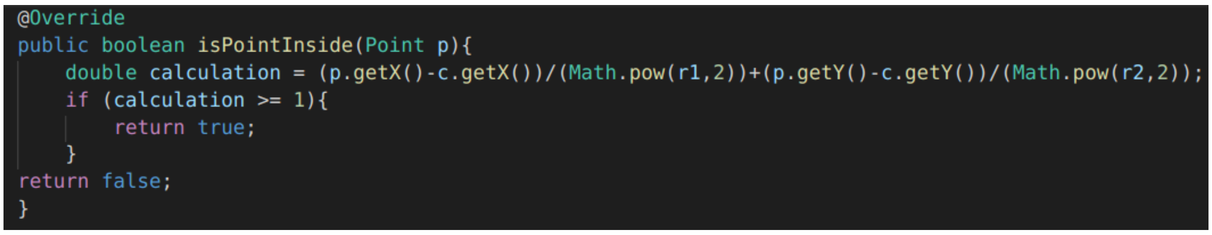
```java
@Override
public void translate(int dx, int dy){
    c.setX(c.getX() + dx);
    c.setY(c.getY() + dy);
}
```

The function translate is used to move the region given a point along the x and y axis.



```java
@Override
public void draw(Graphics g){
    int coordX = c.getX();
    int coordY = c.getY();

    g.setColor(fillColor);
    g.fillOval(coordX, coordY, (int) r1, (int) r2);
    g.setColor( lineColor );
    g.drawOval(coordX, coordY, (int) r1, (int) r2);
}
```

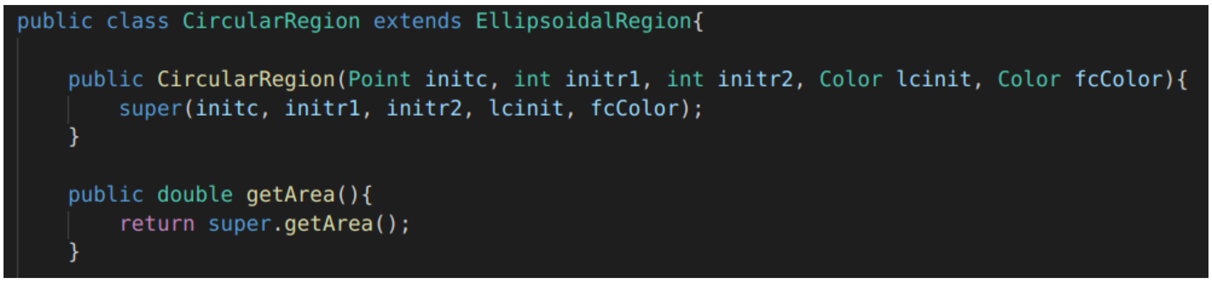
The function draw is used to draw this region.



```java
@Override
public boolean isPointInside(Point p){
    double calculation = (p.getX()-c.getX())/(Math.pow(r1,2))+(p.getY()-c.getY())/(Math.pow(r2,2));
    if (calculation >= 1){
        return true;
    }
return false;
}
```

The IsPointInside function is used to know whether a point we are asking for belongs to the region.

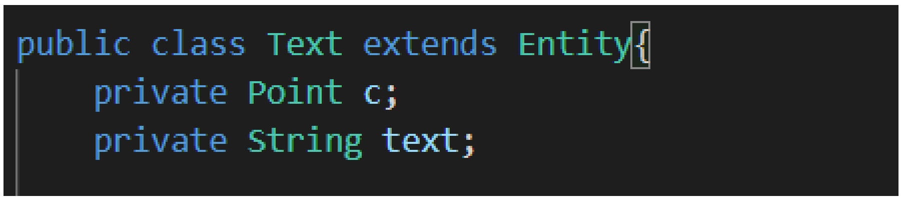
9. Circular Region



```java
public class CircularRegion extends EllipsoidalRegion{

    public CircularRegion(Point initc, int initr1, int initr2, Color lcinit, Color fcColor){
        super(initc, initr1, initr2, lcinit, fcColor);
    }

    public double getArea(){
        return super.getArea();
    }
```

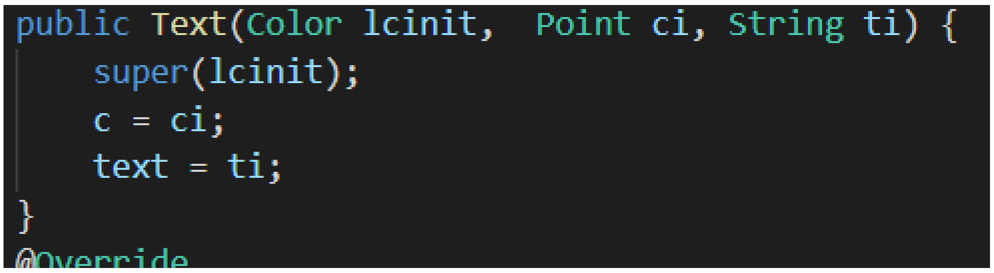
This class inherits from *EllipsoidalRegion*, which at the same time inherits from *Region* which at the same time inherits from *Entity*. That's why in it's construct method we can find the attributes defined in these three classes.
The function getArea uses the function defined in the *EllipsoidalRegion* class.

10. Text



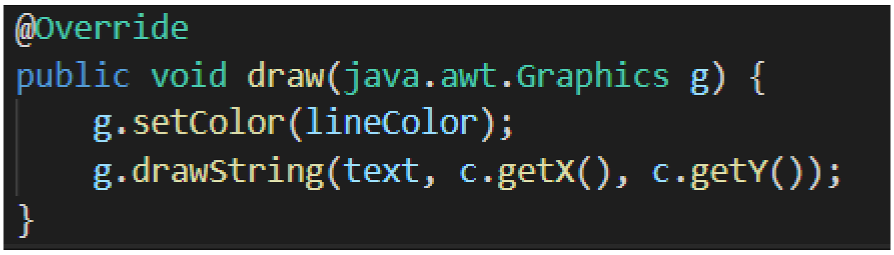
```java
public class Text extends Entity{
    private Point c;
    private String text;
```

These are the two attributes that the class text has. This class inherits from Entity.



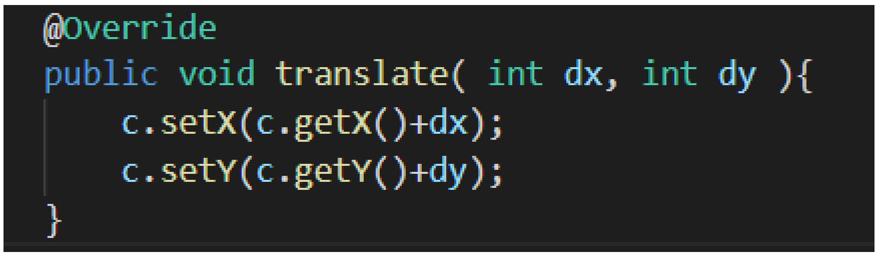
```java
public Text(Color lcinit,  Point ci, String ti) {
    super(lcinit);
    c = ci;
    text = ti;
}
@Override
```

For the constructor, we need to use the super() function in order to set the value of the attribute of the Entity class. The other two attributes are set just normally.



```java
@Override
public void draw(java.awt.Graphics g) {
    g.setColor(lineColor);
    g.drawString(text, c.getX(), c.getY());
}
```

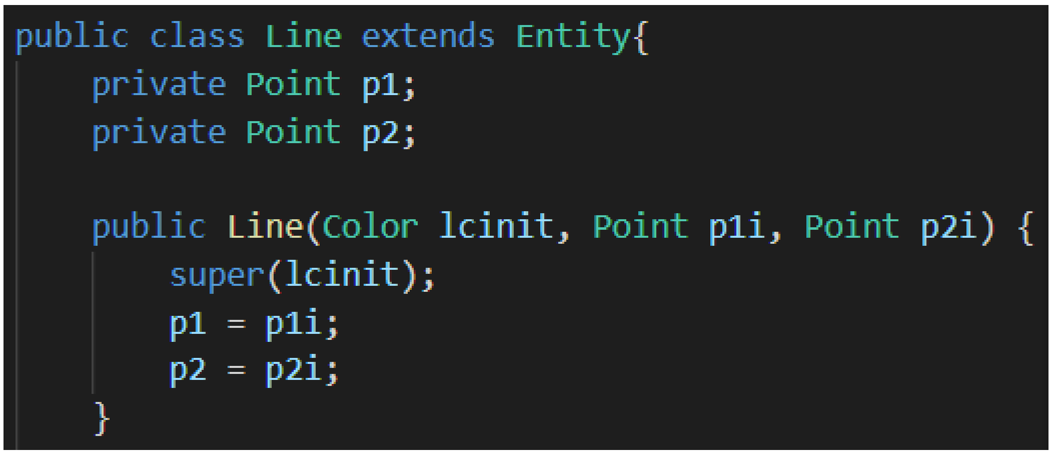
We implement the draw function by using the Graphics class in java. Firstly, we set the color in which the text will be drawn, and then we use the drawString function.



```java
@Override
public void translate( int dx, int dy ){
    c.setX(c.getX()+dx);
    c.setY(c.getY()+dy);
}
```

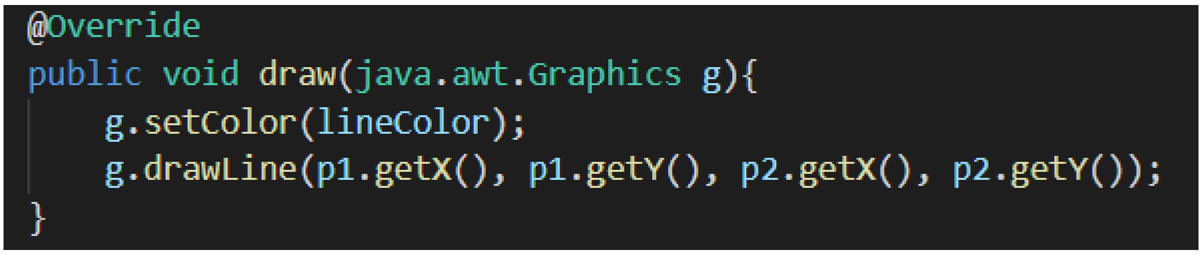
Lastly, we define the translate function, which simply adds the values of the coordinates to the starting point of the text.

11. Line



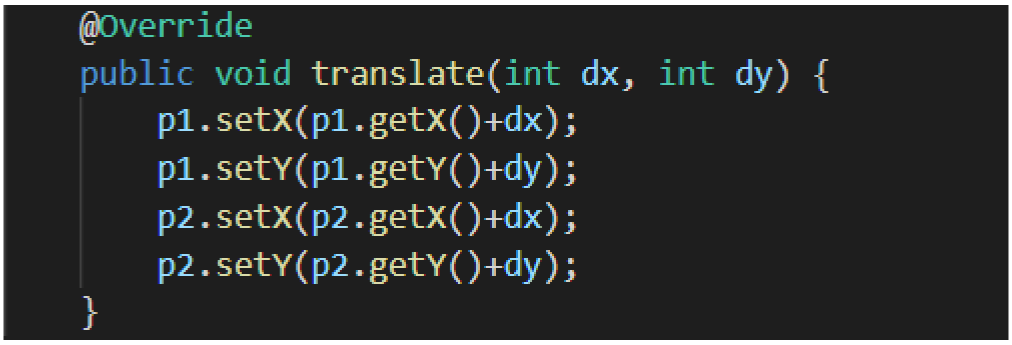
```java
public class Line extends Entity{
    private Point p1;
    private Point p2;

    public Line(Color lcinit, Point p1i, Point p2i) {
        super(lcinit);
        p1 = p1i;
        p2 = p2i;
    }
```

The line class also inherits from Entity. Its attributes are just two points, the starting and the ending points of the line. We also need to add a color parameter in the constructor in order to build the entity by using the super keyword.



```java
@Override
public void draw(java.awt.Graphics g){
    g.setColor(lineColor);
    g.drawLine(p1.getX(), p1.getY(), p2.getX(), p2.getY());
}
```

For the draw function, we simply set the color and call the drawLine function in graphics, passing as parameters the coordinates of the two points.



```java
    @Override
    public void translate(int dx, int dy) {
        p1.setX(p1.getX()+dx);
        p1.setY(p1.getY()+dy);
        p2.setX(p2.getX()+dx);
        p2.setY(p2.getY()+dy);
    }
```

Lastly, the translate function adds the coordinates of the movement to the coordinates of the two points, and sets them to become the current coordinates of the aforementioned points.

Pablo Fernández
Clara López
S101, P101

Conclusion

To sum up, we are satisfied with the work done. It was hard to start at the beginning but at the end we finally got to do the code. In the end, we are happy with the work we have done.