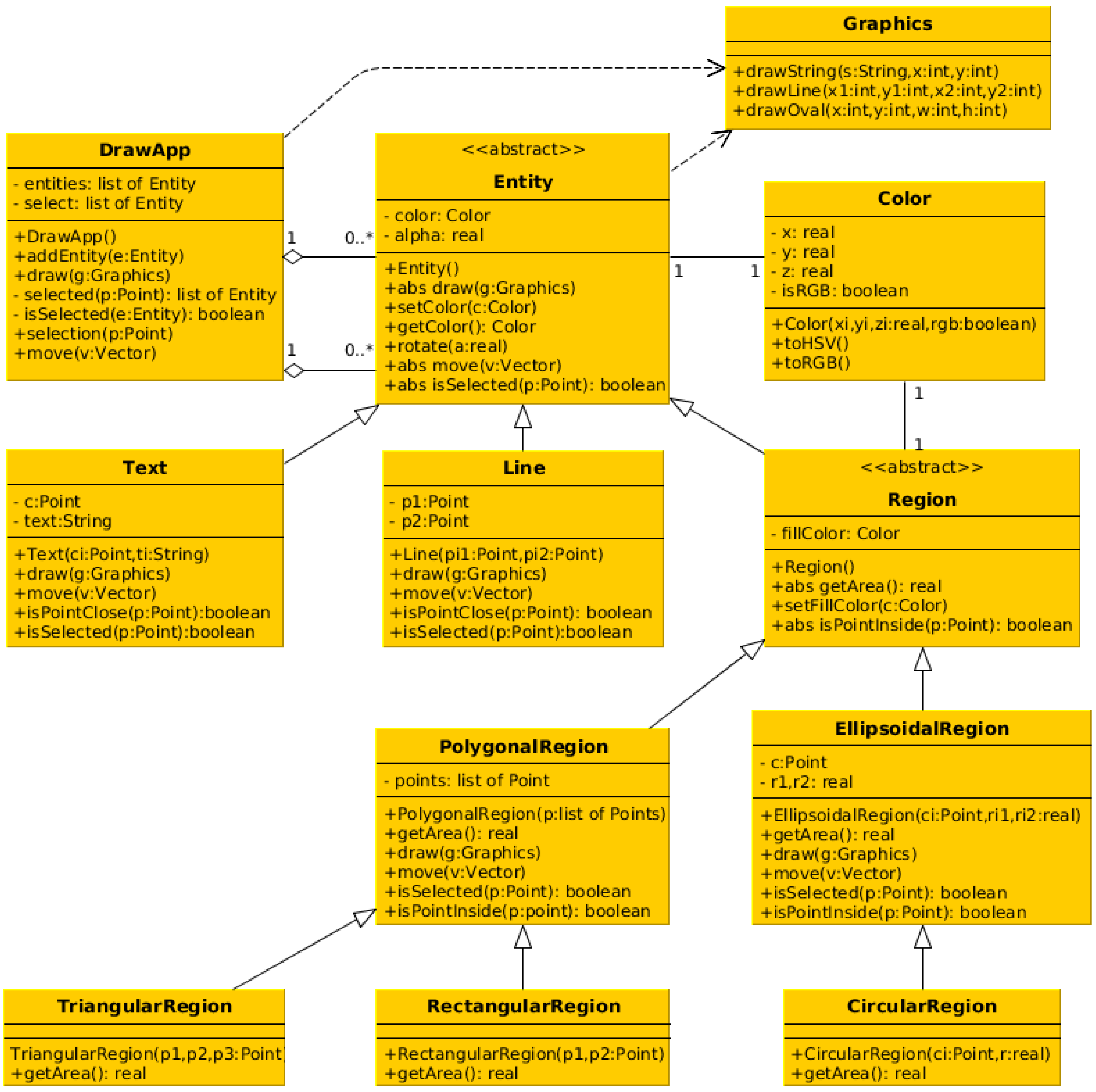
# Lab exercise 4: Implementing a design based on inheritance
# 24292-Object Oriented Programming

## 1 Introduction

The aim of this lab session is to implement the design from Seminar 4 to draw and move regions. In particular, this requires implementing the corresponding operations on geometrical shapes. The session is mandatory and you have to deliver the source code of the Java project and a document describing the implementation decisions and details.

You are free to implement your own design and create your own graphical interface. However, you may instead choose to implement the sample solution to Seminar 4 that appears in the following figure:



In the Aula Global you will also find Java code that includes a graphical interface as well as the abstract Entity class. Note that the DrawPanel class in the code corresponds to the DrawApp class in the design, and the method `translate( int dx, int dy )` in DrawPanel plays the role of the method `move` in DrawApp.

To simplify, we will only implement some of the proposed operations on regions:

- `move/translate`, to translate a region a given distance along the $x$ and $y$ axes,
- `isPointInside`, to determine whether a given point lies inside a region.

In other words, it is not necessary to implement the method for rotating regions.

## 2 Points and vectors

To represent points you can simply use the Point class from Lab Session 2. It is also convenient to implement a Vector class, since we can efficiently determine whether a given point lies inside a convex polygon by computing the cross-product of vectors. Just like a two-dimensional point $p = (p_x, p_y)$, a two-dimensional vector $v = (v_x, v_y)$ is described by two coordinates. The cross-product between two vectors $u$ and $v$ is defined as

$$u \times v = u_x v_y - u_y v_x.$$

It is also convenient to add two new methods to the Point class:

- `move/translate`, to translate a point a given distance along the $x$ and $y$ axes,
- `difference`, to compute the difference between two points (i.e. a vector).

Given two points $p$ and $q$, the difference between the two points is a vector given by

$$p - q = (p_x - q_x, p_y - q_y).$$

## 3 Entities and regions

The purpose of the Entity class is to represent objects that can be drawn on the screen or moved. However, all concrete objects will be instantiated from a subclass of Entity. Hence it is convenient to define Entity as an abstract class. The ability to draw and move objects are represented using the abstract methods `draw` and `translate`. Since all entities have an associated line color, the Entity class includes an attribute representing the line color. Java already defines a Color class (in the library `java.awt`). To specify a concrete color, there exist constants Color.RED, Color.BLUE, Color.GREEN, etc., each of which is an instance of the Color class.

The purpose of the Region class is to represent regions. However, all regions have a concrete shape (polygon, ellipse, etc.), represented by one of the subclasses of Region. Hence Region can also be defined as an abstract class. For each type of region, it is possible to check whether a given point lies inside a region of that type. Hence the method `isPointInside` is common to all subclasses of Region, and can be defined as an abstract method of Region. Since we want to fill regions as we draw them, the Region class includes an attribute representing the fill color. Optionally, you can also include a method for computing the area.

To implement the class PolygonalRegion, you can again use the class with the same name from Lab Session 2. However, PolygonalRegion should now inherit from Region (in case it did not already do so). It is necessary to add the two methods `move/translate` and `isPointInside`. To fill a region with a different color than its boundary, it is also necessary to modify the `draw` method. Recall that the `draw` methods has an argument `g` which is an instance of the Graphics class in the `java.awt` library of Java. Apart from methods `drawPolygon`, `drawOval`, etc. the Graphics class also includes a set of corresponding methods `fillPolygon`, `fillOval`, etc. For the boundary of the polygon to show we first have to fill the polygon using one color, then draw it using another color, as shown in the following piece of code:

```
g.setColor( fillColor );
g.fillPolygon( ... );
g.setColor( lineColor );
g.drawPolygon( ... );
```

To determine whether a point $p$ is inside a polygon, we are first going to assume that the polygon is *convex* (just as we did when we implemented the area of a polygon in Lab Session 2). We are then going to treat the sides of the polygon as vectors, as shown in Figure 1 on the next page.

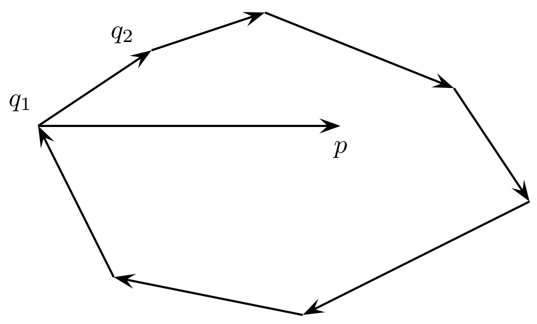


Figure 1: Determine whether a point $p$ is inside a convex polygon.

The point $p$ is inside of a convex polygon if the *sign* of the cross-product $(q_2 - q_1) \times (p - q_1)$ is the same for all pairs of consecutive points $q_1$ and $q_2$ of the polygon. To obtain the vectors $(q_2 - q_1)$ and $(p - q_1)$ you can simply use the new method `difference` of the Point class. Recall that the sign of a number $n$ is $+1$ if $n$ is positive and $-1$ if $n$ is negative.

You also have to implement the new class EllipsoidalRegion which, just like PolygonalRegion, should contain methods `draw`, `move/translate` and `isPointInside`. The draw method can be implemented using the `drawOval` and `fillOval` methods of Graphics. A point $p$ is inside an ellipse precisely when the following inequality holds:

$$\frac{(p_x - c_x)^2}{a^2} + \frac{(p_y - c_y)^2}{b^2} \leq 1,$$

where $a$ and $b$ are the two axes of the ellipse and $c$ is the center point of the ellipse (cf. Figure 2).

Finally, implement the three classes TriangularRegion, RectangularRegion and CircularRegion. These classes inherit most methods from their parent classes, and should be relatively easy to implement.

## 4 Graphical interface

Once all regions have been implemented you can use the class EntityDrawer to launch the graphical application. Define a new class with a main method for testing, and create an instance of EntityDrawer. Note that EntityDrawer has a method `addDrawable` that adds an instance of Entity to the list of drawables. Create several entities and add them to the graphical application using the `addDrawable` method.

If everything works as expected, the entities should be drawn on the screen when the application window launches. By selecting distances $dx$ and $dy$ along the $x$ and $y$ axes and pressing `Move`, all entities should be translated the given distances.

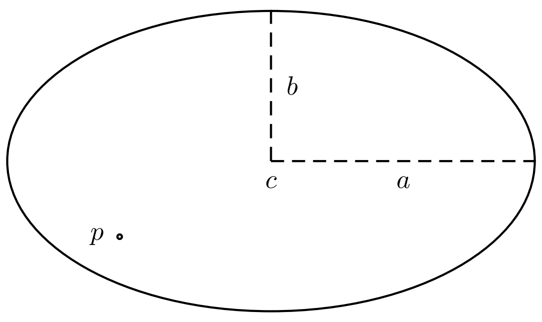


Figure 2: Determine whether a point $p$ is inside an ellipse.

## 5 Optional: implement the selection tool

Currently the selection tool has not been implemented. Optionally, you can implement the selection tool yourself. To do so you have to modify the DrawPanel class as follows:

1. Import the Java library `java.awt.event`.
2. Change the class header to "`public class DrawPanel extends JPanel implements MouseListener`".
3. Include a second attribute "`protected LinkedList< Entity > selection`".
4. Add the following line to the constructor:

   ```
   addMouseListener( this );
   ```

5. Override the five abstract methods of MouseListener in DrawPanel. All except the first method can be empty (i.e. brackets with no code inside).

   ```
   public void mousePressed(MouseEvent e)
   public void mouseReleased(MouseEvent e)
   public void mouseEntered(MouseEvent e)
   public void mouseExited(MouseEvent e)
   public void mouseClicked(MouseEvent e)
   ```

6. In the method `mousePressed` the idea is to select a subset of regions. The $(x, y)$-coordinates of the mouse event are extracted using method calls `e.getX()` and `e.getY()`. Then you have to call the method `isPointInside` for all regions in the list of entities. Add the selected regions to the new list `selection`.
7. Finally modify the method `translate` such that only the entities in the list `selection` are moved.

## 6 Document

Do not forget to include a document that describes the development of your program. The guidelines for the document appear in the handout for Lab Session 1.