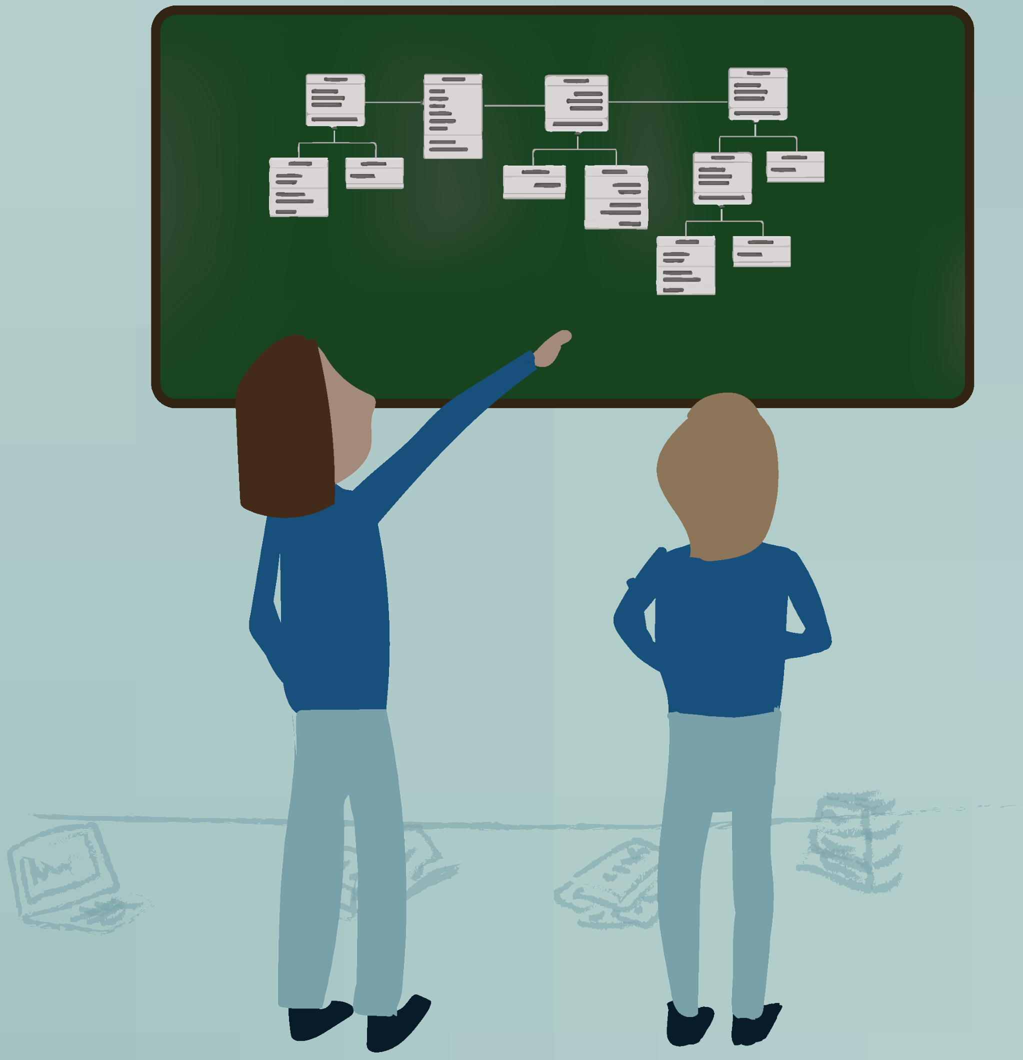
Software Development

Portfolio 1

Spring 2022



Mursal Hosseini *Petr Boska Nylander*

Number of characters with spaces: 10.827

1

1.1 Introduction

In the previous semester we were collaborating with an IT company who wanted to find a solution to their time registration issues. The company uses an older version of Timelog, which is the system where the employee registers their working hours, sick days, holidays etc. Furthermore, uses the Team Manager the system to make invoices to their customers. The company had many issues with the system, but the main issue was the lack of integration between Timelog and their internal accounting system. The company also has some customers who desire a more detailed invoicing on the time consumption, but the requirement from their customers cannot easily be supported by the current time registering system.

In the first part of this assignment, we will analyse the current time registration system the company uses, by first showing how the employee and team manager use the time registration system. Here we are using Use Case diagram, which shows the purpose of the system Timelog and its action.

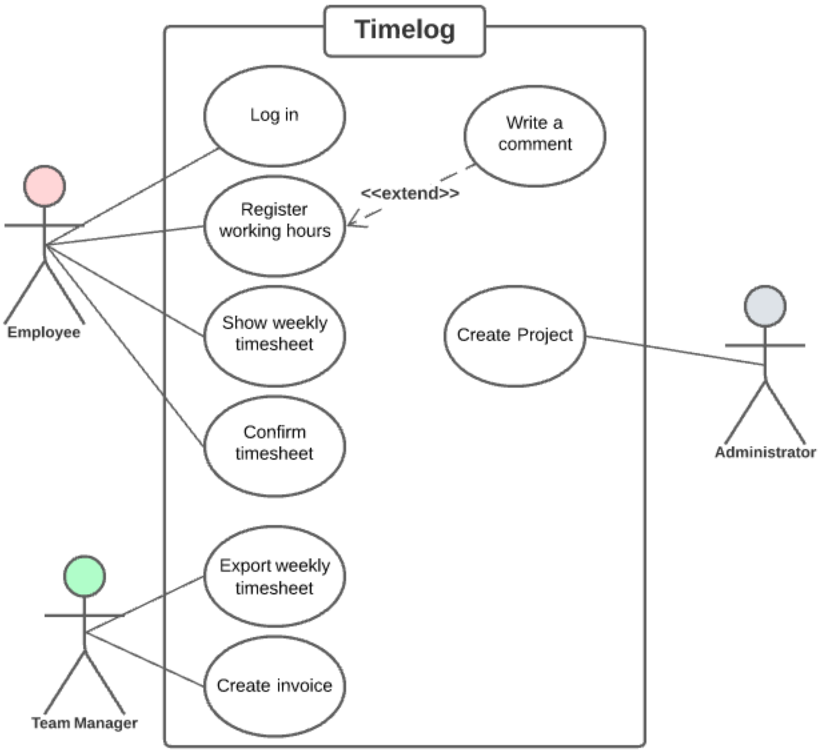


Figure 1: Use Case diagram over the time register system Timelog

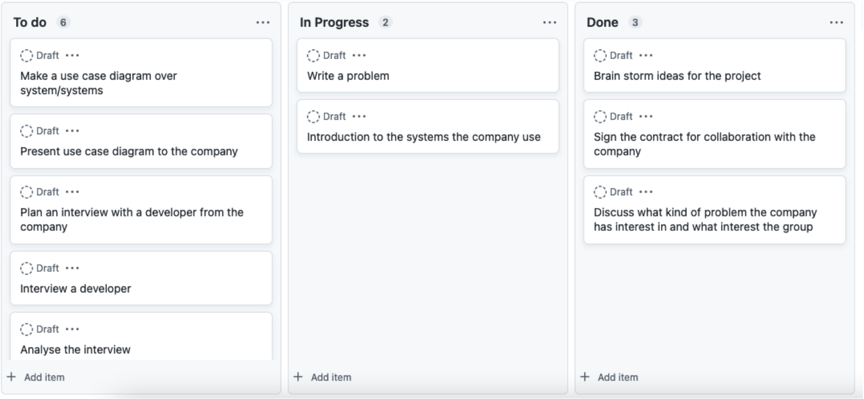
In our Use Case diagram, we have two primary actors – the Employee and the Team Manager who is the user of Timelog and needed to interact with the system. We also have the Administrator, who is the secondary actor, because the only time they interact with Timelog is when they need to create a project. The Employee also has the option to write a comment when registering their working hours, this will be needed if the employee has worked on different projects and needs to document which project their working hours are used on.

The main objective of the project is to find out how Timelog can be optimized, so it is clear for the Team Manager what every work hour is used on and on which project specifically.

1.2 Development environment

1.2.1 Kanban

We are using GitHub which provides an opportunity to create a project with a built-in Kanban board where we have added some tasks that need to be done throughout the project.



To do 6

Draft ···
Make a use case diagram over system/systems

Draft ···
Present use case diagram to the company

Draft ···
Plan an interview with a developer from the company

Draft ···
Interview a developer

Draft ···
Analyse the interview

+ Add item

In Progress 2

Draft ···
Write a problem

Draft ···
Introduction to the systems the company use

+ Add item

Done 3

Draft ···
Brain storm ideas for the project

Draft ···
Sign the contract for collaboration with the company

Draft ···
Discuss what kind of problem the company has interest in and what interest the group

+ Add item

Figure 2: Kanban board from GitHub project

1.2.2 Programming language

For development, we have chosen to program in JAVA as we already have some experience from the course *Essential programming* last semester. JAVA is also a universal and an object-oriented programming language which can be implemented on several platforms.

1.2.3 Version control

For Version control we have chosen to use GitHub, so we can easily share and keep track of our code between the group. We have not yet set up the repository and the branching structure since we are still in the planning stages of our project.

2 Part 2 Shapes implementation

The following section will describe the and break down the Shape class, into a single step, so it can be consequently translated into the UML diagram and implemented and tested.

2.1 Shapes Abstract Class

As it is a requirement for this assignment, the Shape class should be implemented as an abstract class, which includes methods such as:

1. Returning center of shape
2. Area of shape
3. Circumference of shape
4. Is point inside?

These following methods will be declared as abstract methods, since shape can't be mathematically defined, but can define the methods, which can be inherited from this abstract class, for the concrete classes as Circle, Triangle, Rectangle. Another reason these methods are abstract is, that every concrete shape will implement these methods its own way. The implementation of the Abstract Class Shapes, can be seen in the appendix A

2.1.1 Point class

To construct the concrete shapes (Circle, Rectangle and Triangle), a Point class will need to be defined. The Point class will include the Cartesian coordinates x, y, for specifying the position in the 2D plane. Methods for getting the x and y coordinates (called getters), should also be defined, in order to access the coordinates from other classes and methods.

Method for calculating distance between two points should also be defined. The distance can be calculated from the Euclidean distance:

$$\text{Distance between points} \quad AB = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

Lastly, for displaying the point in the style of $[x, y]$ the toString methods, will be defined. The implementation of the Point Class, can be seen in the appendix B

2.1.2 Circle class

The circle class will consist of the private fields of one center point A, with $[x, y]$ coordinates (using the Point class), radius, see figure 3 and the constant π, for calculating the area and circumference of the circle.

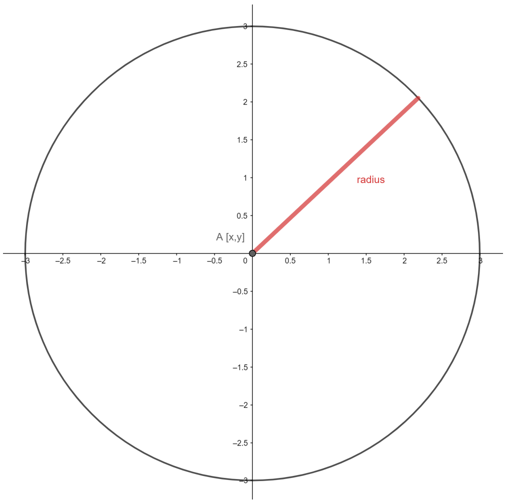


Figure 3: Circle object, with the center point A and radius

At the instantiation of the Circle object, certain restrictions need to be implemented. Namely, the radius of the circle cannot be negative. Therefore, if user enters the negative radius, it will be default set to zero. This restriction will be implemented in the constructor of the Circle class.

Circle will be inheriting the abstract methods, from the Shape class, which will be implemented as following:

1. **Return center**

   As a first, methods for returning of the center of circle needs to be implemented. Since, the center of circle is defined at the instantiating of the Circle object, the point representing the center of circle will be returned.

2. **Area of the circle**

   Another method should return the area of the circle. Area of the circle can be computed as:

$$\text{Circle Area} = \pi \cdot r^2 \tag{2}$$

3. **Circumference of the circle**

   The circumference of the circle will be computed in another method as:

$$\text{Circle Circumference} = 2 \cdot \pi \cdot r \tag{3}$$

4. **Is point inside**

   To find out, whether point will be inside the circle, on the circle or outside of the circle, the equation of the area of the circle could be used to investigate. The area of the circle is following:

$$x^2 + y^2 = r^2 \tag{4}$$

   The x and y coordinates are the coordinates of the circle. When we want to investigate the point with coordinates $[px, py]$, then the equation 4 can be rewritten as follows.

$$(x - px)^2 + (y - py)^2 = r^2 \tag{5}$$

   There are three situations, which can occur. If the point is inside the circle, that would be in a situation, when the left-hand side (LHS) would be less, then the right-hand side (RHS). When, the point is on the circumference of the circle, then the LHS will equal the RHS. And when the point is outside of the circle, the LHS is bigger than the RHS, see figure 4.

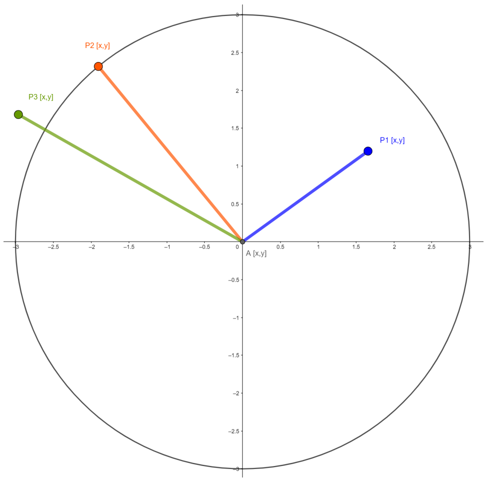


Figure 4: Illustration of the method Is point inside

The implementation of the Circle Class, can be seen in the appendix C

2.1.3 Triangle class

The triangle class will consist of the private fields, with three points A, B, C, where each point is having its $[x, y]$ coordinates, see figure 4. And rest of its methods will be inherited from the abstract class Shapes, which will be implemented as follows:

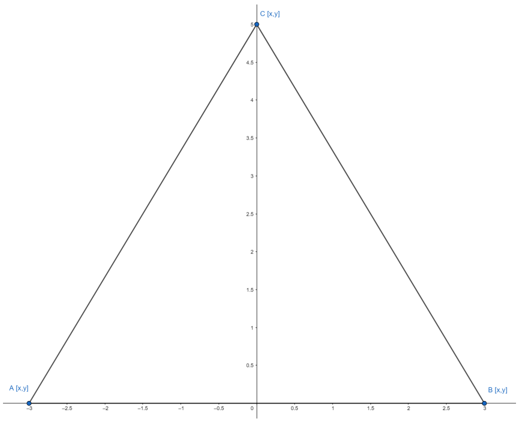


Figure 5: Triangle object, will consist of three points A, B, C

1. **Return center**

   The center of triangle can be found from the equation for centroid of a triangle. The equation looks as follows:

$$\text{Triangle Centroid X coordinate} = \frac{x_1 + x_2 + x_3}{3} \tag{6}$$

$$\text{Triangle Centroid X coordinate} = \frac{y_1 + y_2 + y_3}{3} \tag{7}$$

2. **Area of the triangle**

   The area of the triangle can be calculated, from the following equation:

$$\text{Triangle Area} = \frac{1}{2} \cdot [x_1(y_2 - y_3) + 2(y_3 - y_1) + x_3(y_1 - y_2)] \tag{8}$$

3. **Perimeter of triangle**

   The method for calculating the perimeter of the Triangle, can be done with the earlier defined method in Point class, for calculating distance between two points. It will be used three times, namely for the sides AB, BC, CA. By adding all their sides, the perimeter of the triangle will be found.

$$\begin{aligned} AB &= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \\ BC &= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \\ CA &= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \\ \text{Perimeter of triangle} &= AB + BC + CA \end{aligned} \tag{9}$$

4. **Is point inside**

   For finding out, whether the point is inside the triangle the following methods can be used. If the points are inside the triangle, the equation 8 for the triangle area can be used. As can be seen on figure 6, when the point is inside the triangle, then the triangle can be divided into three smaller triangles connecting the point and two sides from the triangle, namely APB, BPC, APC. Area of these triangles, should give the same area as the area of the original triangle ABC. If the point is outside, the area of the APB, BPC, APC triangles should be larger, than the original one. See figure 7.

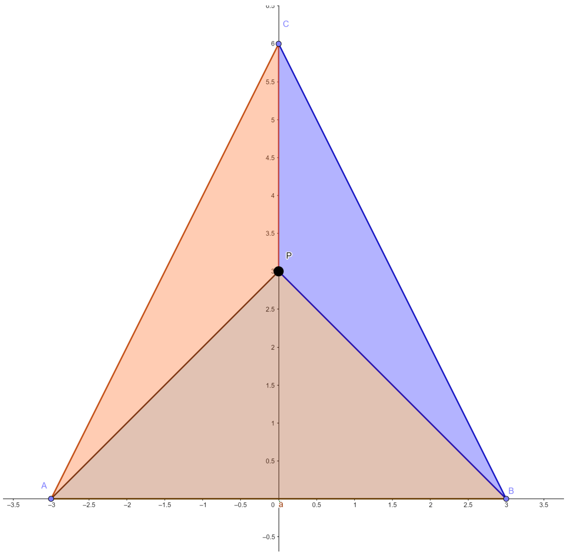


Figure 6: The point inside of the triangle ABC

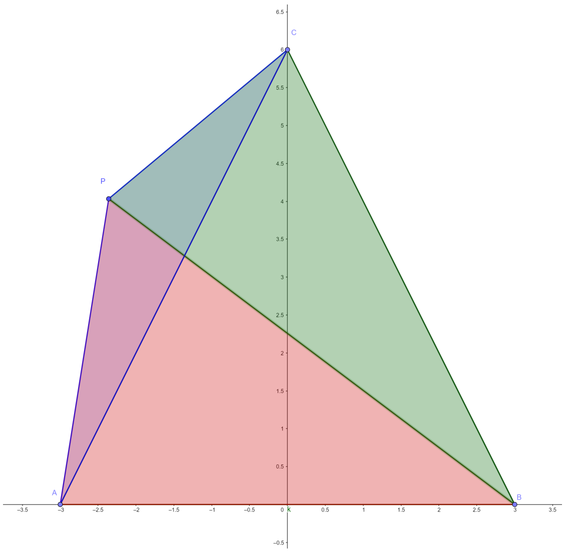


Figure 7: The point is outside of the triangle ABC

The implementation of the Triangle Class, can be seen in the appendix D

2.1.4 Rectangle class

The rectangle class will consist of the private fields, with the four points of A, B, C, D, where each point is having its $[x, y]$, see figure 8.

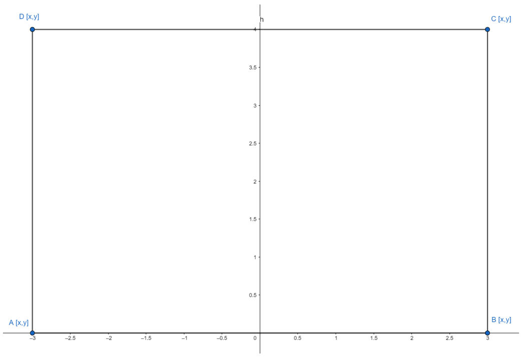


Figure 8: Rectangle object, will consist of four points A, B, C, D

1. **Return center**

   The center of rectangle can be found from the equation for centroid of rectangle. The equation looks as follows:

   $$\text{Rectangle Centroid } x \text{ coordinate} = \frac{x_1 + x_2 + x_3 + x_4}{4} \tag{10}$$

   $$\text{Rectangle Centroid } y \text{ coordinate} = \frac{y_1 + y_2 + y_3 + y_4}{4} \tag{11}$$

2. **Area of the rectangle**

   The area of rectangle, can be found with the equation 8, by dividing the rectangle A, B, C, D into two triangles ABC and ACD, calculating their area and then adding these areas together:

   $$\begin{aligned}&\text{Triangle } ABC = \tfrac{1}{2} \cdot [x_1(y_2 - y_3) + 2(y_3 - y_1) + x_3(y_1 - y_2)]\\&\text{Triangle } ACD = \tfrac{1}{2} \cdot [x_1(y_2 - y_3) + 2(y_3 - y_1) + x_3(y_1 - y_2)]\\&\text{Area of rectangle } ABCD \text{ = } ABC \text{ + } ACD\end{aligned} \tag{12}$$

3. **Perimeter of rectangle**

   The perimeter of the rectangle $ABCD$ can be found by calculating the length of its side AB, representing the width of the rectangle and the length of side BC, representing its length as follows:

   $$\text{Perimeter of rectangle} = 2 \cdot AB + 2 \cdot BC \tag{13}$$

4. **Is point inside**

   The method for finding, whether the point is inside the rectangle $ABCD$ is the same as in the previous section for triangle. With exception, that the rectangle will be divided into four triangles APB, BPC, CPD, APD. Area of these triangles should be the same as the area of the original rectangle $ABCD$, see figure 9. If the point is outside, the area of the triangles APB, BPC, CPD, APD should be larger, that the original one, see figure 10.

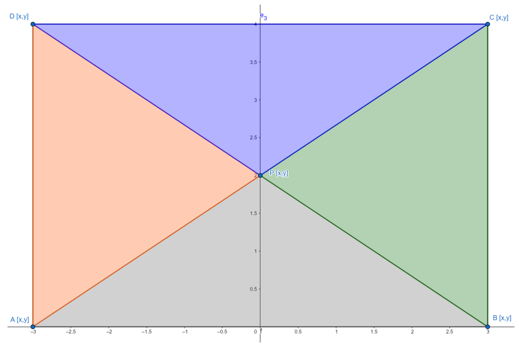


Figure 9: The point is inside of the rectangle $ABCD$

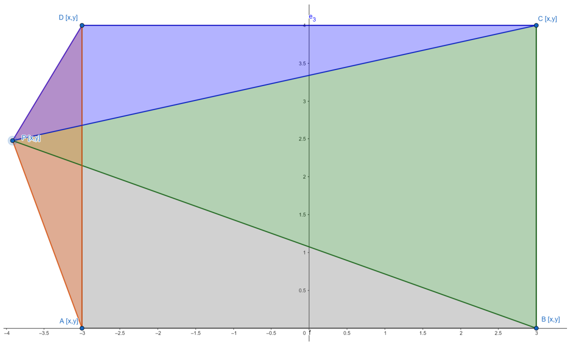


Figure 10: The point is outside of the rectangle $ABCD$

The implementation of the Rectangle Class, can be seen in the appendix E

2.2 UML diagram

In the previous section, we have laid out, how all the methods could be implemented. In this section follows the UML diagram, from which, we have implemented the Shape model. The white arrows, pointing to the Shapes class, signifies inheritance. The classes Triangle, Circle and Rectangle are inheriting the methods from the Abstract Class Shapes. The Points class has a relation with the Triangle, Circle and Rectangle have a relation of composition. These classes are composed of the individual points. Therefore, the black diamonds are signifying the composition. The classes have a different relation to the Point class. Triangle uses 3 points; Circle uses 1 and Rectangle uses 4 points for its construction. See the UML diagram at figure 11.

The methods of Area with the point parameters in the Triangle and Rectangle class, have been added. It is overloading the original Area method, without any parameters. The method has been added after the entire Shape model was implemented to delete repetitive code.

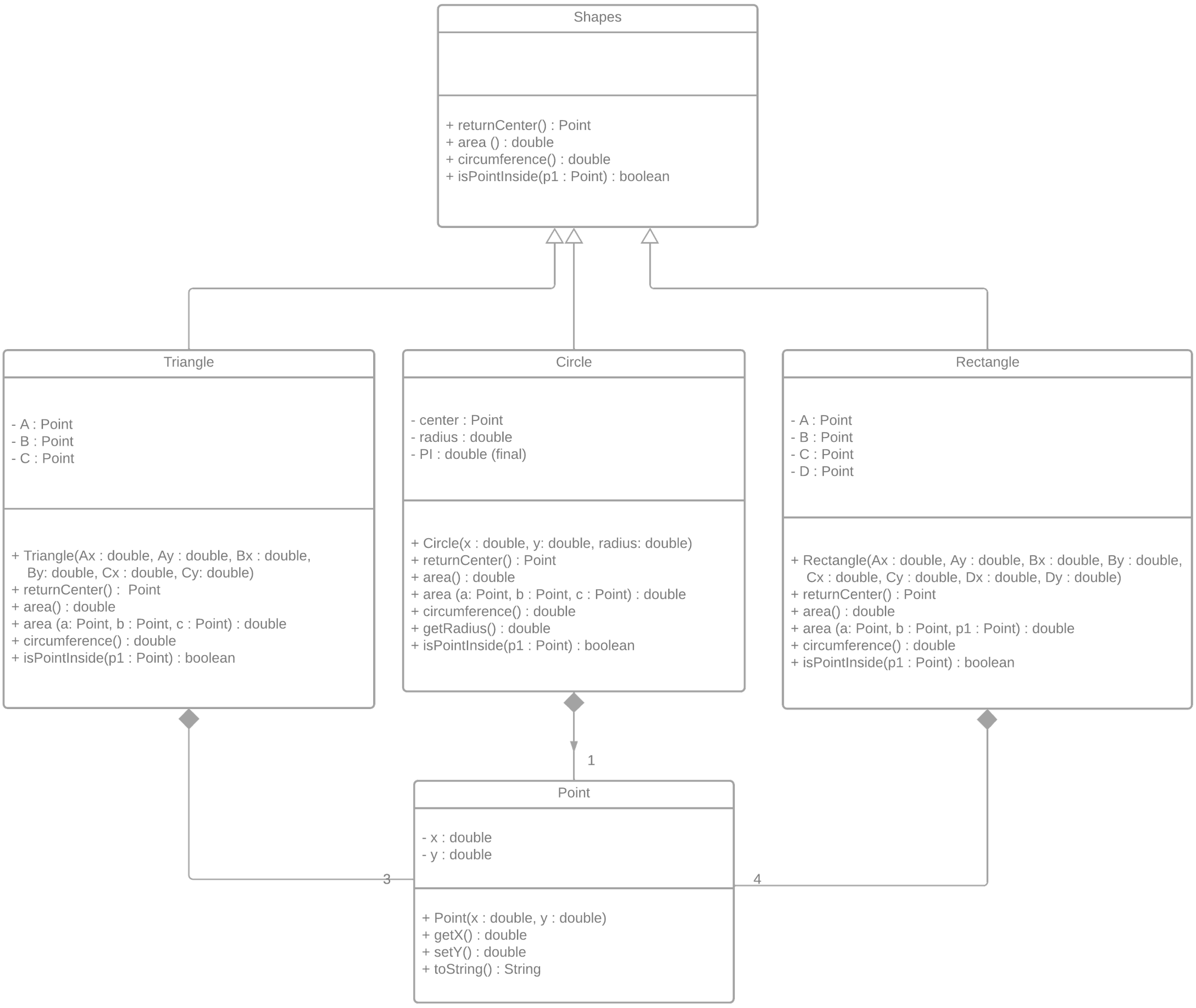

Shapes
+ returnCenter() : Point
+ area () : double
+ circumference() : double
+ isPointInside(p1 : Point) : boolean
Triangle
- A : Point
- B : Point
- C : Point
+ Triangle(Ax : double, Ay : double, Bx : double, By: double, Cx : double, Cy: double)
+ returnCenter() : Point
+ area() : double
+ area (a: Point, b : Point, c : Point) : double
+ circumference() : double
+ isPointInside(p1 : Point) : boolean
Circle
- center : Point
- radius : double
- PI : double (final)
+ Circle(x : double, y: double, radius: double)
+ returnCenter() : Point
+ area() : double
+ area (a: Point, b : Point, c : Point) : double
+ circumference() : double
+ getRadius() : double
+ isPointInside(p1 : Point) : boolean
Rectangle
- A : Point
- B : Point
- C : Point
- D : Point
+ Rectangle(Ax : double, Ay : double, Bx : double, By : double, Cx : double, Cy : double, Dx : double, Dy : double)
+ returnCenter() : Point
+ area() : double
+ area (a: Point, b : Point, p1 : Point) : double
+ circumference() : double
+ isPointInside(p1 : Point) : boolean
1
3
4
Point
- x : double
- y : double
+ Point(x : double, y : double)
+ getX() : double
+ setY() : double
+ toString() : String

Figure 11: The UML diagram for the Shape model

2.3 JUnit testing

2.3.1 Circle

After implementing the circle class, we have subjected its methods to JUnit testing to verify their behaviour. First, we have tested the constructor of the circle class by passing a negative value of radius. Then we tested three positions, where the point will be inside the circle. One point was put directly into the middle of the circle, another one inside at a random place and the last one on the circle's circumference. After that, we have tested the method with the point outside of the circle. After that, we tested the area and circumference methods and distances from the centre to the centre, with two circles in the exact centre, different radiuses and two circles with different centre points. All the testing methods above have passed the testing. The test class for the circle can be seen in appendix F.

2.3.2 Triangle

The testing for the Triangles has been done similarly to the testing for Circle. We have also tested the area and perimeter methods. We have tested whether the point is inside of the triangle or outside. Next, the methods for distances between the centre points of two identical triangles were tested, and then the distances of two triangles with different centres were tested. All the testing methods above have passed the testing. The test class for the triangle can be seen in appendix G.

2.3.3 Rectangle

We have tested the area and perimeter methods for the rectangle class. Afterwards, we tested whether a point is inside the rectangle, on one side of the rectangle and outside of the rectangle. Afterwards, we tested the distances between the centres of identical triangles and distances between the centres of two different rectangles. All the testing methods above have passed the testing. The test class for the triangle can be seen in appendix E.

Appendices

A Abstract Class Shapes

```java
package com.company;

public abstract class Shapes {

    abstract public Point returnCenter();

    abstract public double area();

    abstract public double circumference();

    abstract public boolean isPointInside(Point p1);
}
```

B Class Point

```java
package com.company;

public class Point {
    private double x; // Points x coordinate
    private double y; // Points y coordinate

    public Point(double x, double y) { // Constructor for the Points
        this.x = x;
        this.y = y;
    }
    public double getX() {
        return x;
    } // Getter method for x coordinate
    public double getY() {
        return y;
    } // Getter method for x coordinate

    public static double distanceOfTwoPoints(Point a, Point b) { // distance between two points
        return Math.sqrt(Math.pow(b.getX() - a.getX(), 2) +
                Math.pow(b.getY() - a.getY(),2));
    }
    public String toString() {
        return "[" + x + "," + y+ "]";
    } // Method for printing the point coordinate
}

```

C Class Circle

```java
package com.company;

public class Circle extends Shapes{ // Circle extending the Shape class
    private Point center;
    private double radius;
    private static final double PI = Math.PI;

    public Circle(double x, double y, double r) { // Constructor for circle class
        center = new Point(x, y);
        if (r > 0) {                        // if radius entered to negative value
            this.radius = r;                // it is set to 0
        } else {
            this.radius = 0;
        }
    }
    @Override
    public Point returnCenter() {      // method for returning center for the circle
        return center;
    }

    public double getRadius() {
        return radius;
    }    // getter method for radius

    @Override
    public double area() {
        return PI*radius*radius;
    } // method for calculating area of circle

    @Override
    public double circumference() {
        return 2*PI*radius;
    } // method for calculating circumference

    @Override
    public boolean isPointInside(Point p1) {// method for investigating whether point is inside
        if (Math.pow(p1.getX() -center.getX(),2) + Math.pow(p1.getY() - center.getY(),2) < radius*radius) { // calculation whether point is inside circle
            return true;
        } else if (Math.pow(p1.getX() -center.getX(),2) + Math.pow(p1.getY() - center.getY(),2) == radius*radius) { // calculation for whether point is on circle
            return true;
        }
        return false;       // if point is outside, method returns false
    }
}


```

D Class Triangle

```java
package com.company;

public class Triangle extends Shapes{ // Triangle extending the Shape class
    private Point A;
    private Point B;
    private Point C;

    public Triangle(double Ax, double Ay, double Bx, double By, double Cx, double Cy) { //Constructor taking parameters for 6 x and y coordinates
        this.A = new Point(Ax, Ay);
        this.B = new Point(Bx, By);
        this.C = new Point(Cx, Cy);
    }

    @Override
    public Point returnCenter() {       // method returning the center of triangle
        double centerX = (int) ((A.getX() + B.getX() + C.getX())/3);
        double centerY = (int) ((A.getY() + B.getY() + C.getY())/3);
        Point center = new Point(centerX,centerY);      // creates and returns a new point for center
        return center;
    }
    @Override
    public double area() {              // method returning the area of triangle
        double area =  (((A.getX()*(B.getY()- C.getY()) +
                B.getX()*(C.getY()- A.getY())  +
                C.getX()*(A.getY()- B.getY()))/2));
        return area;
    }

    public double area(Point A, Point B, Point C) {      // overloaded method of the area(), which is taking 3 parameters
        double area = A.getX() * (B.getY() - C.getY()) / 2 +     // in order to eliminate reperitive code when using this method
                B.getX() * (C.getY() - A.getY()) / 2 +       // for finding, whether point is inside or outside triangle
                C.getX() * (A.getY() - B.getY()) / 2;
        return area;
    }
    @Override
    public double circumference() {                  // perimeter of the triangle
        double sideAB = Point.distanceOfTwoPoints(A,B);
        double sideBC = Point.distanceOfTwoPoints(C,B);
        double sideCA = Point.distanceOfTwoPoints(C,A);
        return sideAB + sideBC + sideCA;
    }
    @Override
    public boolean isPointInside(Point p1) {         // method detecting whether the point is inside or not
        double A1 = Math.abs(area(A,B,p1));
        double A2 = Math.abs(area(A, p1, C));
        double A3 = Math.abs(area(A,C,p1));
        double areaWithPoint = A1 + A2 + A3;
        if (areaWithPoint == area()) return true;    // if point inside, return true
        return false;                                // if not, return false
    }
}


```

E Class Rectangle

```java
package com.company;

public class Rectangle extends Shapes{ // Rectangle extending the shape class
    private Point A;
    private Point B;
    private Point C;
    private Point D;

    public Rectangle(double Ax, double Ay, double Bx, double By, double Cx, double Cy, double Dx, double Dy) { // constructor for Rectangle, that takes 8 x and y coordinates
        this.A = new Point(Ax, Ay);
        this.B = new Point(Bx, By);
        this.C = new Point(Cx, Cy);
        this.D = new Point(Dx, Dy);
    }

    @Override
    public Point returnCenter() { // method returning the center of rectangle
        double centerX = ((A.getX() + B.getX() + C.getX() + D.getX())/4);
        double centerY = ((A.getY() + B.getY() + C.getY() + D.getY())/4);
        Point center = new Point(centerX,centerY);
        return center;
    }
    @Override
    public double area() {    // method for returning the area of rectangle
        double A1 = A.getX() * (B.getY() - D.getY()) / 2
                + B.getX() * (D.getY() - A.getY()) / 2
                + D.getX() * (A.getY() - B.getY()) / 2;
        double A2 = D.getX() * (B.getY() - C.getY()) / 2
                + B.getX() * (C.getY() - D.getY()) / 2
                + C.getX() * (D.getY() - B.getY()) / 2;
        return A1 + A2;
    }

    double area(Point A, Point B, Point p1) {                // overloaded method of the area(), which is taking 3 parameters
```

```
        double A1 = A.getX() * (B.getY() - p1.getY()) / 2   // in order to elimitnate repetitive ode when using this method
                + B.getX() * (p1.getY() - A.getY()) / 2     // for finding, whether point is inside or outside rectangle
                + p1.getX() * (A.getY() - B.getY()) / 2;
        return Math.abs(A1);
    }
    @Override
    public double circumference() {                         // perimeter of rectangle
        double length = Point.distanceOfTwoPoints(A,B);
        double width =  Point.distanceOfTwoPoints(B,C);
        return 2*length + 2*width;
    }
    @Override
    public boolean isPointInside(Point p1) {               // method detecting whether the point is inside or not
        double A1 = Math.abs(area(A,D,p1));
        double A2 = Math.abs(area(D,C,p1));
        double A3 = Math.abs(area(C,B,p1));
        double A4 = Math.abs(area(p1,B,A));
        double areaWithPoint = A1+A2+A3+A4;
        if (areaWithPoint == this.area()) return true;      // if point inside, return true
        return false;                                       // if not, return false
    }
}


```

F Test Class Circle

```
package com.company;
import org.junit.jupiter.api.Test;
import static org.junit.Assert.assertEquals;

public class CircleTest {
    @Test
    public void testOfCircleConstructor() { // if negative radius, should be set to 0
        Circle c1 = new Circle(1,1,-1);
        assertEquals(0,c1.getRadius(),.1);
    }
    @Test
    public void testPointInside1() { // Point in the center of circle
        Circle c1 = new Circle(0,0,5);
        Point p1 = new Point(0,0);
        assertEquals(true, c1.isPointInside(p1));
    }
    @Test
    public void testPointInside2() { // Pont inside of circle
        Circle c1 = new Circle(0,0,5);
        Point p1 = new Point(0,1);
        assertEquals(true, c1.isPointInside(p1));
    }
    @Test
    public void testPointInside3() { // Point on the boarder
        Circle c1 = new Circle(0,0,5);
        Point p1 = new Point(0,5);
        assertEquals(true, c1.isPointInside(p1));
    }
    @Test
    public void testPointOutside() { // Point on the boarder
        Circle c1 = new Circle(0,0,5);
        Point p1 = new Point(0,6);
        assertEquals(false,c1.isPointInside(p1));
    }
    @Test
    public void testArea() {
        Circle c1 = new Circle(0,0,5);
        assertEquals(78.5398163, c1.area(),.1);
    }
    @Test
    public void testCircumference() {
        Circle c1 = new Circle(0,0,5);
        assertEquals(31.4159265, c1.circumference(),.1);
    }
    @Test
    public void testDistanceCenterToCenter() { //testing the distance between two circles with the same center
        Circle c1 = new Circle(0,0,5);
        Circle c2 = new Circle(0,0,2);
        assertEquals(0, Point.distanceOfTwoPoints(c1.returnCenter(),c2.returnCenter()),.1);
    }
    @Test
    public void testDistanceCenterToCenter2() { //testing the distance between two circles with the different center
        Circle c1 = new Circle(5,5,5);
        Circle c2 = new Circle(0,0,2);
        assertEquals(7.07, Point.distanceOfTwoPoints(c1.returnCenter(),c2.returnCenter()),.1);
    }
}

```

G Test Class Triangle

```
package com.company;

import org.junit.jupiter.api.Test;

import static org.junit.Assert.*;

public class TriangleTest {
    @Test
    public void testArea() { // test of the area calculation
        Triangle t1 = new Triangle(-3,0,3,0,0,6);
        assertEquals(t1.area(), 18,.1);
    }
    @Test
    public void testCircumference() { // test of the circumference calculation
        Triangle t1 = new Triangle(-3,0,3,0,0,6);
        assertEquals(t1.circumference(), 19.416, .1);
    }
    @Test
    public void isPointInside1() { // test, when point is inside
        Triangle t1 = new Triangle(-3,0,3,0,0,6);
        Point p1 = new Point(0,2);
        assertEquals(true,t1.isPointInside(p1));
    }
    @Test
    public void isPointInside2() { // test, when point is outside
        Triangle t1 = new Triangle(-3,0,3,0,0,5);
        Point p1 = new Point(-10,-10);
        assertFalse(t1.isPointInside(p1));
    }
    @Test
    public void testDistanceCenterToCenter() { //testing the distance between two triangles with the same center
        Triangle t1 = new Triangle(-2,0,2,0,0,3);
        Triangle t2 = new Triangle(-2,0,2,0,0,3);
        assertEquals(0, Point.distanceOfTwoPoints(t1.returnCenter(),t2.returnCenter()),.1);
    }
    @Test
    public void testDistanceCenterToCenter2() { //testing the distance between two rectangles with the different center
        Triangle t1 = new Triangle(-2,0,2,0,0,3);
        Triangle t2 = new Triangle(-4.01,0,4,0,0,9);
        assertEquals(2, Point.distanceOfTwoPoints(t1.returnCenter(),t2.returnCenter()),.1);
    }
}

```

H Test Class Rectangle

```
package com.company;
import org.junit.jupiter.api.Test;
import static org.junit.Assert.*;

public class RectangleTest {
    @Test
    public void areaTest() { // test the area of rectangle
        Rectangle r1 = new Rectangle(-3,0,5,0,5,5,-3,5);
        assertEquals(r1.area(),40,.1);
    }
    @Test
    public void circumferenceTest() { // test the area of rectangle
        Rectangle r1 = new Rectangle(-3,0,5,0,5,5,-3,5);
        assertEquals(r1.circumference(),26,.1);
    }
    @Test
    public void isPointInsideTest1() { // test for point is inside of the rectanglge
        Rectangle r1 = new Rectangle(-3,0,5,0,5,5,-3,5);
        Point p1 = new Point(-2,3);
        assertTrue(r1.isPointInside(p1));
    }
    @Test
    public void isPointInsideTest2() { // point is on the side of rectangle
        Rectangle r1 = new Rectangle(-3,0,5,0,5,5,-3,5);
        Point p1 = new Point(-3,3);
        assertTrue(r1.isPointInside(p1));
    }
    @Test
    public void isPointInsideTest3() { // Test for point outside of rectangle
        Rectangle r1 = new Rectangle(-3,0,5,0,5,5,-3,5);
        Point p1 = new Point(-4,3);
        assertFalse(r1.isPointInside(p1));
    }
    @Test
    public void testDistanceCenterToCenter() { //testing the distance between two rectangles with the same center
        Rectangle r1 = new Rectangle(-2,-5,2,-4,2,4,-2,4);
        Rectangle r2 = new Rectangle(-2,-5,2,-4,2,4,-2,4);
        assertEquals(0, Point.distanceOfTwoPoints(r1.returnCenter(),r2.returnCenter()),.1);
    }
    @Test
    public void testDistanceCenterToCenter2() { //testing the distance between two rectangles with the different center
        Rectangle r1 = new Rectangle(-2,-5,2,-4,2,4,-2,4);
        Rectangle r2 = new Rectangle(-7.5,-1.5,-6,-1.5,-6,1,-7.5,1);
        assertEquals(6.75, Point.distanceOfTwoPoints(r1.returnCenter(),r2.returnCenter()),.1);
    }
}

```