Due date: Tuesday, March 08, at the beginning of the lab.

1 About this Assignment

1.1 Objectives of the assignment

The objective of this assignment is for you to

- start using conditional statements;
- improve your shape classes by adding them functions allowing to decide whether a shape contains an other, whether two shapes overlap, etc.

Read the text of this assignment very carefully before rushing to type code. Read it completely through once. Do not stop at the first unclear sentence you encounter; sometimes things are explained and detailed a bit later. Then you should start all over again. This time stop to ask questions when a point remains fuzzy, code your solution, and move on to the next section.

1.2 Handouts

This week, the single handout for the lab assignment is a pdf (Acrobat) file containing the present document.

2 Point Ordering Convention

2.1 Counterclockwise ordering

In Computer Graphics most algorithms and implementations of these algorithms in libraries such as OpenGL, Direct3D, Java3D, etc. assume that the points of a polygon are listed in counterclockwise order. The reason for this is that if this convention is respected, it is possible to compute the correct value of the normal vector of the polygon, that is, a direction that is (1) perpendicular to the polygon; and (2) indicates the “up” direction for the polygon. This notion of the “up” side is really important because it affects the way light will be reflected by the polygon, and therefore the way it will be rendered in a viewing window. When you take into account the fact that most 3D scenes (all of them in video games) are made up of a large number of small polygons, you will easily understand why it is so important to compute the correct normal vector for each polygon, and therefore why we must care about the ordering of a polygon’s vertices.
2.2 How we will handle that

You cannot prevent the user of your application (remember: that’s someone who clicks around and enters data in input dialog windows’ boxes) from entering coordinates that correspond to a clockwise ordering of the vertices. You cannot prevent a user of your class (this one is another programmer who writes an application that uses the classes you implemented) from invoking the constructor of your Triangle class by passing clockwise-ordered points either.

There are basically four ways to address these problems:

1. If you are wearing your application developer hat on, you can check if input data are valid before you create an object. You do this by invoking a static method of your class.

2. If you are wearing your class developer hat on, then you could

   (a) Refuse to create an instance object of your class using invalid data. We will learn how to do this in a few weeks, using the Exception classes of Java.

   (b) Re-arrange the data so that they are now valid. This is, however, not always possible. Even when it is possible, it is rarely a good idea. In the case of your Triangle class, you could reorder the vertices so that they are now listed counterclockwise, but since you don’t know what the user of your Triangle class uses it for, you cannot know if you reordering won’t have nasty side effects on his/her code. You could argue that if that programmer is clueless enough to use your class improperly, then he/she deserves such nasty side effects, but this is not considered acceptable behavior in software development.

   (c) Accept the data as they are sent to your class, and then

       • provide methods allowing the user to check the validity of the data stored in a given object;
       • implement variations of the correct algorithms to handle invalid data

       As you can imagine, this can represent a lot of additional work if the class is complicated.

   In this assignment, we will only implement Solution (1). Additionally, Solution (2.c) is offered for extra credit [15 pts] to those of you with lots of free time on their hands.

3 Shape Validity Methods

In this section, we will be adding methods that check whether a shape’s data are valid.

3.1 The Circle class

3.1.1 Two more constructors

First, you should add two more constructors to your class. These correspond to some of the circle calculations you did in Lab 03, but I simply forgot to ask for them when I wrote the class specifications for Lab 04:
Circle(Point2D.Double theCenter, Point2D.Double thePoint)
This constructor receives a reference to the center of the circle and to another point that lies on
the circle. Of course, you know that you should store a copy of the center. As to the other
point, you don’t need to store it, just use it to compute the radius of the circle.

Circle(double xc, double yc, double xp, double yp)
This constructor receives the coordinates of the circle’s center and of another point that lies on
the circle. Here again, the second point should only be used to calculate the radius: you
should not store its coordinates in your class.

3.1.2 Validity test functions
This functions should be static, so that you can invoke it from your application before you try to
create an invalid Circle. The data sent to create a Circle object can be invalid if

• One of the data points sent is invalid (null reference);
• The radius of the circle is negative

You should write a validity test method matching each of the constructors for this class. I will
give the list and formats of these functions for this class. For the other two classes, I will only give
the criteria for validity and you will have to come up with the correct test functions on your own.
For the Circle class, the data validity test functions are

• public static boolean isValidData(Point2D.Double theCenter, double
  theRadius)
• public static boolean isValidData(Point2D.Double theCenter, Point2D.Double
  thePoint)
• public static boolean isValidData(double xc, double yc, double
  theRadius)

Note that there is no need to check the sign of a radius that you calculate in your class by using a
distance function since you already know that the value returned by the sqrt method is always
positive. The test should only be performed on the input data.

3.2 The Triangle class
3.2.1 Validity tests
Just as with the Circle class, a first source of data invalidity for the Triangle class is the
passing of null references to Point2D.Double objects. This is something that you will need
to check for.

The other way the data of a Triangle object can be invalid is if the points are not listed
counterclockwise. Luckily, there is a simple way to do that. For this we can use a different formula
for the area of a triangle than the one we saw in Lab 03. If the coordinates of the vertices of our triangle are \(x_1, y_1\), \(x_2, y_2\), and \(x_3, y_3\), then the area of the triangle is

\[
A = \frac{1}{2} ((x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)).
\]

(1)

How does this relate to the ordering of the points, you ask. If the points are listed counterclockwise, then \(A\) is positive (or zero), otherwise it is negative.

Now you should not have any problem implementing data validity test methods for this class.

### 3.2.2 Triangle degeneracy test

Even if the triangle is listed counterclockwise, there is still a special case to handle: that of a triangle whose points are colinear (lie on a same straight line). This type of triangle is called degenerate, i.e., it is not a “real” triangle. This corresponds of course to the case when the area of the triangle is zero. You should implement the following instance method of the \(\text{Triangle}\) class:

```java
public boolean isDegenerate()
```

This method should return \(true\) if the triangle is degenerate, \(false\) otherwise.

### 3.3 The \(\text{Rectangle211}\) class

#### 3.3.1 Validity tests

Just as with the \(\text{Circle}\) and \(\text{Triangle}\) classes, a first source of data invalidity for the \(\text{Rectangle211}\) class is the passing of \(null\) references to \(\text{Point2D.Double}\) objects. This is something that you will need to check for.

Other ways the data of a \(\text{Rectangle211}\) object can be invalid are the following

- the points are not listed counterclockwise;
- consecutive sides are not perpendicular to each other;
- opposite sides are not of equal length

The good news is that we only need to verify the first two conditions because the third one is just a consequence.

**Counterclockwise ordering**

How do you verify that 4 points are listed counterclockwise to form a quadrilateral (we are not sure that it is a rectangle yet)? You can decompose your rectangle into two triangles and then answer two separate questions regarding the ordering of these triangles. You will have to figure out on your own what the two triangles should be.
**Condition for a right angle**  
If \( v_1 = (x_1, y_1) \), \( v_2 = (x_2, y_2) \), and \( v_3 = (x_3, y_3) \) are three consecutive vertices of our contour, in order to have a right angle in \( v_2 \) (edge \( v_1v_2 \) is perpendicular to edge \( v_2v_3 \)), the coordinates must verify the following condition:

\[
(x_2 - x_1)(x_3 - x_2) + (y_2 - y_1)(y_3 - y_2) = 0.
\]

### 3.3.2 Rectangle degeneracy test

A rectangle is degenerate if one of its side has length zero. In this case the “rectangle” is in fact reduced to a line segment. You should implement the following instance method of the `Rectangle211` class:

```java
public boolean isDegenerate()
```

This method should return `true` if the rectangle is degenerate, `false` otherwise.

### 4 Point-shape Relation

#### 4.0.3 Point inside a circle

This one is really easy: A point is inside the circle if its distance to the circle’s center is less than the circle’s radius:

```java
public boolean isInside(Point2D.Double p)
```

The method should return `true` if the point is inside the circle, and `false` otherwise.

#### 4.0.4 Point inside a triangle

If the vertices of your triangle are listed counterclockwise, then a point is *inside* the triangle if it lies on the *left* side of each of the three edges of the triangles. Now, how do you determine if a point lies on the left side of an edge? Figure it out by yourself. It has something to do with counterclockwise ordering again. . . .

Having figured out how to answer this question, you should be able to implement the following method for the `Triangle` class:

```java
public boolean isInside(Point2D.Double p)
```

The method should return `true` if the point is inside the triangle, and `false` otherwise.

#### 4.0.5 Point inside a rectangle

This one is really easy. I should not have to tell you how to do it:

```java
public boolean isInside(Point2D.Double p)
```

The method should return `true` if the point is inside the rectangle, and `false` otherwise.
5 “Contains” Relation Between Shapes

Next, I would like to be able to check if one shape contains another. For this, you will have to send a reference to a shape object to another shape object and ask “do you contain that shape?” To do this you will need for the first shape to access the data (vertices for example) of the other shape. You must use the access methods of your classes to do so.

This is not because Java will not let you do it: unless you did something special to restrict access to the instance variables of your shape classes, they are wide open for any outside use. The reason why I tell you not to use them is because it is a bad programming practice:

- accessing directly the instance variables of another class allows to go modify the data “behind the back” of the developer of that class. For example, you could turn a valid triangle into an invalid one. This would lead to problems if the developer of that class (of course, so far you are the only developer involved, but we are talking of general good behavior here) only checks once if the data is valid, when the object is created. A properly designed class should offer access methods to its data. That way, any change of the values of the data can be validated within the class.

- The developer of a class could change at any time the type and name of the instance variables of the class. Seen from the outside, it should not make any difference to a user of the class whether you store the center of a Circle object as two doubles named x and y, as a Point2D.Double object named center, or as a Point2D.Double object named myCenter as long as the access and calculation methods keep the same names and parameter signatures. This is why I gave you precise specifications for these methods, and none at all for the instance variables: what you decide to name the latter, and even their type is really your own business (of course, some choices are better than others, and we will give you feedback on that).

5.1 Triangle or Rectangle inside a Circle

A circle triangle (resp. a rectangle) is inside a circle if the three vertices of the triangle (resp. the four vertices of the rectangle) are inside the circle. You should have no problem implementing the following methods for the Circle class.

- public boolean isInside(Triangle tri)

- public boolean isInside(Rectangle rect)

5.2 Triangle inside a Rectangle or Rectangle inside a Triangle

These tests are as easy as the previous ones. A triangle is inside a rectangle if all its vertices are inside the rectangle. A rectangle is inside a triangle if all its vertices are inside the triangle. You should add the following methods.
• In the Triangle class:
  public boolean isInside(Rectangle211 rect) returns true if rect is inside this Triangle.

• In the Rectangle211 class:
  public boolean isInside(Triangle rect) returns true if rect is inside this Rectangle211.

5.3 Circle inside a Rectangle211

This one is so simple that I won’t even give you any hint or directions for it\(^2\). Just implement the following method in the Rectangle211 class

public boolean isInside(Circle circ)

this method should return true if the circle is entirely inside this rectangle, and false otherwise.

5.4 Circle inside a Triangle

In order for a circle to be completely inside a triangle, two conditions must be verified:

• The circle’s center must be inside the triangle;
• the distance between the circle’s center and each of the three sides of the triangle must be greater than the circle’s radius.

You already know how to determine whether a point is inside a triangle, so we only need new code for the second test to implement the method

public boolean isInside(Circle circ)

for the Triangle class. This method should return true if the circle is entirely inside the triangle and false otherwise.

6 Overlapping Shapes

Initially I had planned to ask you to implement methods that determine whether two shapes overlap (are partially contained by the other), but I think that you will have enough to do with the isInside methods. Maybe we will return to the overlapping problem in a few weeks, this time with polygons that have larger numbers of sides.

7 How You Will Be Evaluated

7.1 Point distribution

The maximum number of points is 100, but extra points could be awarded for excellent aspects of the project or report. The point distribution for this assignment is as follows:

\(^2\)Actually, I will, sort of. There are two ways of doing this. The first one will only work for a rectangle whose sides are horizontal and vertical. This is the method “so simple I won’t bother explain it.” The other method does for the rectangle what we will do next for the triangle. This method will work for any convex polygon.
Execution evaluation
End-of-session evaluation 10 pts
Execution of the project handed in 30 pts
Source Code
Identifier names 10 pts
Good indentation and general readability 10 pts
Judicious comments well positioned in the code 15 pts
Report
List of the program’s main variables and their use 5 pts
Discussion 10 pts
General quality of the writing and presentation 10 pts

7.2 Various point penalties
Hopefully we won’t have to apply many of these:

Project left accessible on the workstation -5 pts
Project folder incomplete or not properly cleaned up -5 pts
Report file missing from the project folder -5 pts

Late penalties
Printed copy of the report, 1 day late -5 pts
Project folder (uploaded to EnVision server), per day late -10%

If you submit a project late, then it is your responsibility to notify the TA (with CC. to the instructor) that the project is finally available for download on the EnVision server. If you fail to do so, then the “late penalty clock” will keep ticking until the TA gets around to checking your folder on the EnVision server and notices your project. Unless specifically asked to do so, do not mail your project folder as an attachment.