Due date: Tuesday, March 03, 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 developing your own classes;
- get used to passing references to objects as parameters to your methods;
- learn to use the Java online documentation to write better code.

Read very carefully the text of this assignment 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 this fourth lab assignment is a pdf (Acrobat) file containing the present document.

This assignment is a bit special because it is due two weeks from now. We won’t have a lab next Tuesday, but make sure to check the web site next Monday because a complement to this assignment will be posted there giving you some directions for the implementation of some of the methods.

1.3 Classes at long last

If while working on Lab 03 you spent a large fraction of your time complaining about all the code duplication and redundant coding, then you are ready for this assignment. If not, then maybe you could go back add triangles and squares to last week’s program and return to us when you begin to feel the pain.

Obviously, since we have been talking about classes and objects since the beginning of the semester, you must feel that we should have created new classes for our various geometrical shapes, and then created instances of these classes. This is the program for today.
2 Our Geometric Shape Classes, Version 1

We are going to define a

2.1 The Point2D.Double class and the Java API

All our shapes are defined by one or more points. Each point has a set of $x$ and $y$ coordinates. We could naturally store these coordinates directly in our shape classes, but Java already offers some classes for points. The Point class has been a part of Java right from the beginning (JDK 1.0), but it can only store int coordinates. Since many of you already used floating point numbers for their coordinates, we might as well use the Point2D.Double class that was introduced in Java 1.2.

The first thing to do when you encounter a new class is to have a look at the documentation of the Java API (Application Programming Interface). This documentation is available on-line. The latest version of the Java Development Kit is JDK 1.5, also commonly called “Java 5.0”. You can access this documentation at the following URL:

http://java.sun.com/j2se/1.5.0/docs/api/index.html

You can also download a complete set of the documentation to install on your computer. This is highly recommended if you haven’t got always access to the Internet while developing Java code. You cannot be expected to know by heart the details of the hundreds of classes Java is made of, but we will (increasingly so) expect you to be able to find your way through the huge API’s documentation to locate the classes you need for your assignments, and to understand how to use them properly. Of course we will give you a lot of guidance and hints as you learn your way around the JDK.

Now, looking at the documentation for the Point2D.Double class, we learn a few important things. First, it is a part of the java.awt.geom package. This means that you are going to need to add the appropriate import statement before the definition of any class that uses Point2D.Double.

Another important thing we learn is the parameters expected by the various constructors of the class. For example, you can create a new Point2D.Double object by sending along the $x$ and $y$ coordinates of the point:

```
Point2D.Double pt1 = new Point2D.Double(2.5, 5.0);
```

will create a new Point2D.Double object whose coordinates are $x = 2.5$ and $y = 5.0$.

Finally, we see that the Point2D.Double class offers a large number of methods that perform useful calculations. A particularly important set of methods allow us to set or get the value of an object’s instance variables. These methods are called access methods, and you will need to implement such methods for your new classes.

Make sure that you understand how to use the Point2D.Double class. Create a few such objects, and then set and get the values of their coordinate before you move on to the next subsection.
2.2 The Circle class

A circle is defined by its center and radius. We will implement circles in a class named Circle. We will write the source code for this class in a file named Circle.java. When you ask Eclipse to create a new Java class, you should make sure that the class does not contain a main method (if you went too fast and forgot to deselect the box, just delete the method in the newly created class). When implementing your Circle class, you will store the center as a Point2D.Double and the radius as a double. Then you will implement the following methods:

- ** Constructors 
  - Circle(double x, double y, double theRadius) 
    This constructor initializes a new circle whose center has the coordinates specified by x and y and whose radius is theRadius.
  - Circle(Point2D.Double theCenter, double theRadius) 
    This constructor receives a reference to a Point2D.Double object defining the center of the circle, and the value of the radius. Be very careful that you don’t want to store a copy of the reference to the Point2D.Double object but a new Point2D.Double object that has the same coordinates.

- ** Access methods 
  - void setCenter(double x, double y) 
    This method allows to change the coordinates of the circle’s center.
  - double getCenterX( ) 
    This method returns the x coordinates of the circle’s center.
  - double getCenterY( ) 
    This method returns the y coordinates of the circle’s center.
  - void setRadius(double theRadius) 
    This method allows the user of your class to change the radius of the circle.
  - double getRadius( ) 
    This method returns the radius of the circle.

- ** Calculation methods 
  - double getPerimeter( ) 
    This method returns the circle’s perimeter.
  - double getArea( ) 
    This method returns the circle’s area.

Note that the above list of methods is not restrictive. In particular, to implement the calculations methods, you may need to create new methods to handle repetitive operations. Feel free to do so.

Having implemented your Circle class, you should be able to create new Circle object in the main method of your main class and use these objects to do the same kind of calculations that you were doing last week, but now in in a much less cumbersome way.
2.3 The Triangle class

A triangle is defined by its three vertices. Naturally, we will be using the `Point2D.Double` class for the storage of the vertices when we implement our `Triangle` class. Your class should offer the following methods (again, feel free to add more if you want, but the methods listed here must be implemented, and must be so as specified):

- **Constructors**
  - `Triangle(double x1, double y1, double x2, double y2, double x3, double y3)`
    This constructor initializes a new triangle whose vertices have the \( x \) and \( y \) coordinates given in the list of parameters.
  - `Triangle(Point2D.Double pt1, Point2D.Double pt2, Point2D.Double pt3)`
    This constructor receives reference to three `Point2D.Double` objects defining the vertices of the triangle. Be very careful that you don’t want to store copies of the references to these `Point2D.Double` objects but new `Point2D.Double` objects that have the same coordinates.

- **Access methods**
  - `void setVertex1(double x, double y)`
    This method allows to change the coordinates of the triangle’s first vertex.
  - `void setVertex2(double x, double y)`
    This method allows to change the coordinates of the triangle’s second vertex.
  - `void setVertex3(double x, double y)`
    This method allows to change the coordinates of the triangle’s third vertex.
  - `Point2D.Double getVertex1()`
    This method returns a copy (not a reference to) of the triangle’s first vertex.
  - `Point2D.Double getVertex2()`
    This method returns a copy (not a reference to) of the triangle’s second vertex.
  - `Point2D.Double getVertex3()`
    This method returns a copy (not a reference to) of the triangle’s third vertex.
  - `double getX1()`
    This method returns the \( x \) coordinates of the triangle’s first vertex.
  - `double getY1()`
    This method returns the \( y \) coordinates of the triangle’s first vertex.
  - `double getX2()`
    This method returns the \( x \) coordinates of the triangle’s second vertex.
  - `double getY2()`
    This method returns the \( y \) coordinates of the triangle’s second vertex.
- double getX3( )
  This method returns the $x$ coordinates of the triangle’s third vertex.

- double getY3( )
  This method returns the $y$ coordinates of the triangle’s third vertex.

**Calculation methods**

- double getPerimeter( )
  This method returns the triangle’s perimeter.

- double getArea( )
  This method returns the triangle’s area.

Please note that I am very specific on the way your methods should be named and on the number, type, and order of the parameters they take (what is called the *parameter signature* of the method), but not at all on the way you should name your variables. This is because I am specifying the *public API* of your class, that is, the way a user of your class$^1$ should be able to use it. A user of your class should be able to call your class’s methods confidently, know that they are conform to the specifications.

### 2.4 The Rectangle211 class

Java already has a `Rectangle` class, so we are going to implement instead our own `Rectangle211` class. Generally, it is preferable to use Java’s classes rather than trying to develop your own that only differ in a few minor aspects$^2$. This is what we did with the `Point2D.Double` class. For the rectangle, however, we are going to ignore this (good) rule because there are a few important elements of class design that I want to illustrate.

Your class should offer the following methods (again, feel free to add more if you want, but the methods listed here must be implemented, and must be so as specified):

**Constructors**

- Rectangle211(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4)
  This constructor initializes a new square whose vertices have the $x$ and $y$ coordinates given in the list of parameters.

- Rectangle211(Point2D.Double theUpperLeft, Point2D.Double theLowerRight)
  This constructor receives reference to two `Point2D.Double` objects defining the upper-left and lower-right corners of the rectangle.

$^1$Users of your class are other programmers, who may want to use your `Triangle` class to develop their own applications. By opposition, the users of your `application` are typically non-programmers who just click in boxes, enter values, and expect to get results.

$^2$Later this semester we will see how encapsulation and inheritance allow us to add to an existing class.
- Rectangle211(Point2D.Double theCenter, double width double height)
  This constructor receives reference to a Point2D.Double object defining the center of the rectangle, and the value of its width and height.

- **Access methods**
  - Point2D.Double getUpperLeft( )
    This method returns a copy (not a reference to) of the rectangle’s upper-left corner.
  - Point2D.Double getUpperRight( )
    This method returns a copy (not a reference to) of the rectangle’s upper-right corner.
  - Point2D.Double getLowerLeft( )
    This method returns a copy (not a reference to) of the rectangle’s lower-left corner.
  - Point2D.Double getLowerRight( )
    This method returns a copy (not a reference to) of the rectangle’s lower-right corner.
  - void setWidth(double theWidth)
    This method assigns a new value to the width of your rectangle. The upper-left corner of the rectangle is kept fixed and the rectangle gets resized to its new width.
  - double getWidth( )
    This method returns the value to the width of your rectangle.
  - void setHeight(double theHeight)
    This method assigns a new value to the height of your rectangle. The upper-left corner of the rectangle is kept fixed and the rectangle gets resized to its new height.
  - double getHeight( )
    This method returns the value to the height of your rectangle.

- **Calculation methods**
  - double getPerimeter( )
    This method returns the rectangle’s perimeter.
  - double getArea( )
    This method returns the rectangle’s area.

Here you have to make several important design decisions regarding the storage of the state of your rectangle, that is, the definition of the instance variables of your class. Are you going to store the center, width and height of your rectangle? the upper-left corner, width and height? Will you rather store the upper-left and lower-right points? Would you rather store the four corners? Maybe the four corners and the center, width, and height? The latter choice would be called a redundant storage strategy.

These different strategies have their respective strengths and weaknesses. The advantage of redundant storage is that all information is always there under the form you need. The price to pay for this is that you need to keep all these instance variables updated and synchronized whenever
one of them changes. For example, whenever the `setHeight` method is invoked, the height, center, lower-left, and lower-right corners of the rectangle have to be updated. By opposition you can chose to store as little information as possible, and then calculate the value of the other variables when a method requiring these values is called.

Again, both strategies are valid, as long as you implement them properly.

3 Use your new classes in your application

Your application should demonstrate the use of the various methods you have implemented.

4 Our Geometric Shape Classes, Version 2

In this section we will start adding new methods to our classes. Specifications for these will be posted next week. We will implement method that allow us to

- calculate the bounding box of a circle or triangle
- determine whether a point lies within a shape object
- determine whether two shape objects intersect
- determine whether one shape object contains an other

5 What to Hand in

5.1 End-of-session evaluation

You are not expected to complete the assignment by the end of the lab session, but you are definitely expected to have done some work during that session. Try to use the lab session to make sure you understand everything about the assignment. Ask questions; try things; ask more questions.

You should not leave the lab before your work has been evaluated. This first evaluation is worth 10 pts out of 100 for the complete assignment. If you leave before you have been evaluated, these points are lost with no chance of a later evaluation.

5.2 Your project

A folder containing your source files and your report.
5.3 Printed copy of the report

You should hand in a printed copy of your report at the beginning of the next lab session. If your report is not ready at the beginning of the session, a late penalty (never less than one day, or 10%) will be applied. There is no use typing the report during the lab since the penalty is the same whether you return the report at the end of the lab or the next day at the beginning of the class.

6 How You Will Be Evaluated

6.1 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:

<table>
<thead>
<tr>
<th>Execution evaluation</th>
<th>Source Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>End-of-session evaluation</td>
<td>Identifier names</td>
</tr>
<tr>
<td>Execution of the project handed in</td>
<td>Good indentation and general readability</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source Code</th>
<th>Execution evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier names</td>
<td>10 pts</td>
</tr>
<tr>
<td>Good indentation and general readability</td>
<td>10 pts</td>
</tr>
<tr>
<td>Judicious comments well positioned in the code</td>
<td>15 pts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Report</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>List of the program’s main variables and their use</td>
<td>5 pts</td>
</tr>
<tr>
<td>Discussion</td>
<td>10 pts</td>
</tr>
<tr>
<td>General quality of the writing and presentation</td>
<td>10 pts</td>
</tr>
</tbody>
</table>

6.2 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 |

<table>
<thead>
<tr>
<th>Late penalties</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Printed copy of the report, 1 day late</td>
<td>-5 pts</td>
</tr>
<tr>
<td>Project folder (uploaded to EnVision server), per day late</td>
<td>-10%</td>
</tr>
</tbody>
</table>

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.