Homework 5

Due 9am, December 11. No extensions!

What to hand in. For each problem, hand in a copy of your java source code and a script file showing input/output from typical program runs. If you are working on Windows, cut and paste your input/output into a word file.

Problem 1 (Intermediate): As shown in Figure 1 below, rectangles may be defined by the (x,y) coordinates of corner points that are diagonally opposite.

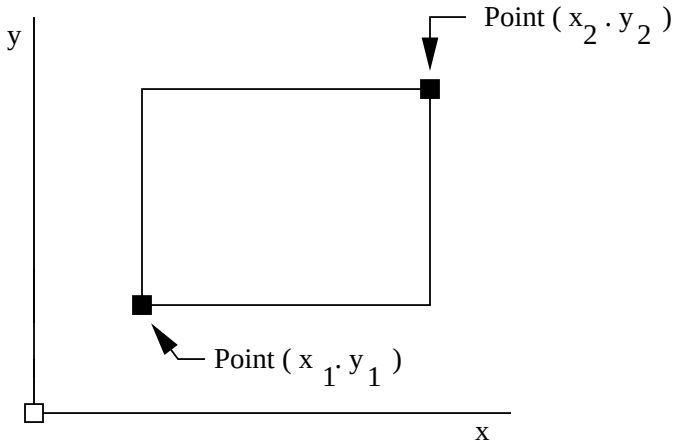

Figure 1: Definition of a rectangle via diagonally opposite corner points

With this definition in place, the following script of code is a very basic implementation of a class for creating and working with rectangle objects.

```
/*
 *  ========================================================================
 *  Rectangle.java : A library of methods for creating and managing rectangles
 *
 *  double      area() -- returns the area of a rectangle
 *  double perimeter() -- returns the perimeter of a rectangle
 *
 *  Written By : Mark Austin                                  November 2009
 *  ========================================================================
 */
```

```
import java.lang.Math;

public class Rectangle {
   protected double dX1, dY1;  // Coordinate (x,y) for corner 1....
   protected double dX2, dY2;  // Coordinate (x,y) for corner 2....

   // Constructor methods ....

   public Rectangle() {}

   public Rectangle( double dX1, double dY1, double dX2, double dY2 ) {
      this.dX1     = dX1; this.dY1     = dY1;
      this.dX2     = dX2; this.dY2     = dY2;
   }

   // Convert rectangle details to a string ...

   public String toString() {
          return "Rectangle: Corner 1: (x,y) = " + "(" + dX1 + "," + dY1 + ")\n" +
                 "           Corner 2: (x,y) = " + "(" + dX2 + "," + dY2 + ")\n";
   }

   // ===================================
   // Compute rectangle area and perimeter
   // ===================================

   public double area() {
       return Math.abs( (dX2-dX1)*(dY2-dY1) );
   }

   public double perimeter() {
       return 2.0*Math.abs(dX2-dX1) + 2.0*Math.abs( dY2-dY1 );
   }

   // Exercise methods in the Rectangle class ....

   public static void main ( String args[] ) {

      System.out.println("Rectangle test program     ");
      System.out.println("==========================");

      // Setup and print details of a small rectangle....

      Rectangle rA = new Rectangle( 1.0, 1.0, 3.0, 4.0 );
      System.out.println( rA.toString() );

      // Print perimeter and area of the small rectangle....

      System.out.println( "Perimeter = " + rA.perimeter() );
      System.out.println( "Area      = " + rA.area() );
   }
}
```

The script of program output is as follows:

```
Script started on Sat Nov 27 10:17:29 2009
prompt >>
prompt >> java Rectangle
Rectangle test program
==========================
Rectangle: Corner 1: (x,y) = (1.0,1.0)
           Corner 2: (x,y) = (3.0,4.0)

Perimeter = 10.0
Area      = 6.0
prompt >> exit
Script done on Sat Nov 27 10:17:39 2009
```

The Rectangle class has methods to create objects (i.e, Rectangle), convert the details of a rectangle object into a string format (i.e., toString), and compute the rectangle area and perimeter (i.e., area() and permieter(), respectively). The implementation uses two pairs of doubles (dX1, dY1) and (dX2, dY2) to define the corner points.

Suppose that, instead, the corner points are defined via a Vertes class, where

```
public class Vertex {
    protected double dX, double dY

    ..... details of constructors and other methods removed ...
}
```

The appropriate modification for Rectangle is:

```
public class Rectangle {
   protected Vertex vertex1;  // First corner point....
   protected Vertex vertex2;  // Second corner point....

   ..... details rectangle removed ....
}
```

Fill in the missing details (i.e., constructors and toString() method) of class Vertex. Modify the code in Rectangle to use Vertex class. The resulting program should have essentially has the same functionality as the original version of Rectangle. **Hint.** Your implementation should make use of the toString() method in Vertex.

Problem 2 (Moderate): There are lots of problems in engineering where the position of point needs to be evaluated with respect to a shape. Evaluation procedures can be phrased in terms

of questions. For example, is the point inside (or outside) the shape?; Does the point lie on the boundary of the shape?; Does the point lie above/below the shape? Does the point lie to the left or right of the shape?; How far is the point from the perimeter?

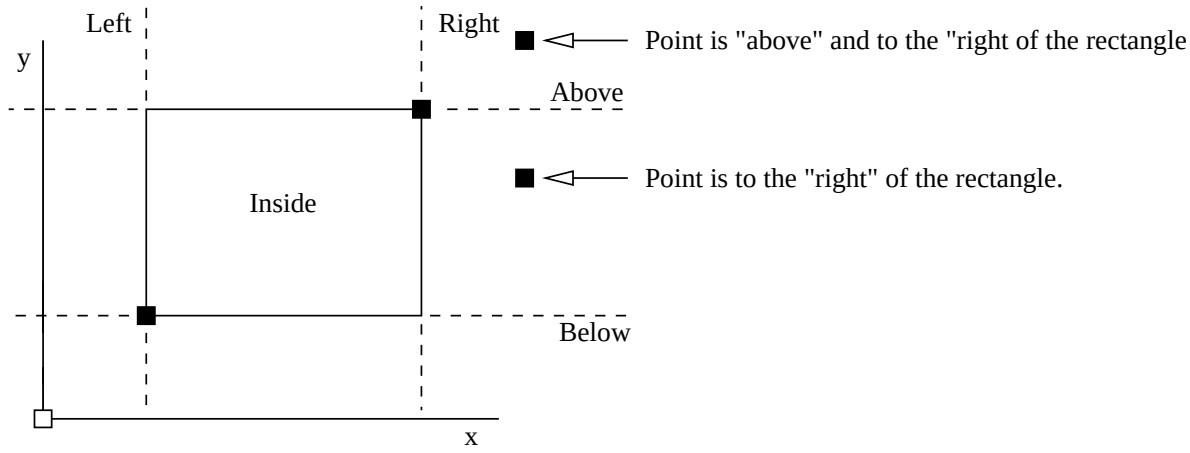

Figure 2: Classification of an (x,y) cooordinate relative to a rectangle

Figure 2 illustrates these ideas for one of the simplest cases possible, one point and a rectangle. Extend the functionality of the Rectangle class so that the possition of a point can be evaluated with respect to a specific rectangle object.

The appropriate method declarations are as follows:

```
public boolean isInside ( Vertex v ) { ... }
public boolean isOutside ( Vertex v ) { ... }
public boolean isOnPerimeter ( Vertex v ) { ... }
public boolean isAbove ( Vertex v ) { ... }
public boolean isBelow  ( Vertex v ) { ... }
public boolean isLeft ( Vertex v ) { ... }
public boolean isRight ( Vertex v ) { ... }
```

If the (x,y) coordinates of a vertex are inside a particular rectangle, then `isInside ()` should return `true.` Otherwise, it should return `false`. From Figure 2 is should evident that some points will result in multiple methods returning true. For example, points in the top right-hand side of the coordinate system will be outside, to the right, and above the rectangle.

Fill in the details of each method, and then develop a test program to exercise the procedures. Perhaps the most straight forward way of doing this is to write an extensive set of tests in the main() method for class Rectangle.

Note. It may be convenient to simply hand in Problems 1 and 2 and as single program.

Problem 3 (Intermediate): This problem will give you practice at using the DataArray class to read, compute, and print the statistics of data collected from a structural engineering experiment in which strain measurements are recorded over an extended period of time.

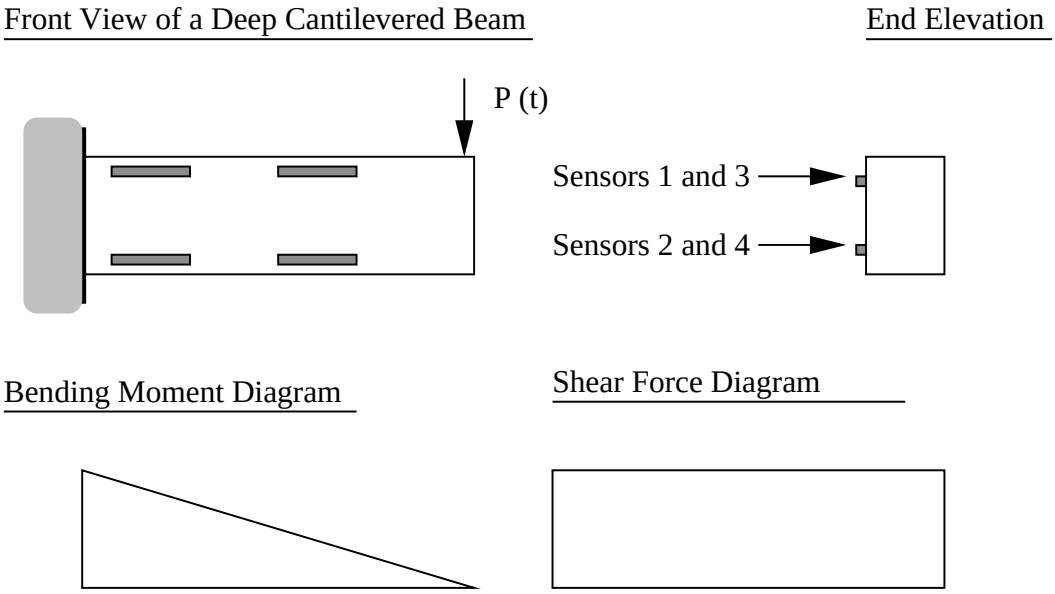

Figure 3: Experiment on flexure and shear of a deep cantilever beam.

Figure 3 shows front and end elevation views of an experimental setup to test the flexural (bending) and shear strength of a deep cantilever beam. The beam is "deep" because the aspect ratio (beam length)/(beam depth) is less than 10 – all of the simplified theory on beam behavior that you will learn as an undergraduate does not apply!

Four sensors are attached to the center of the beam as shown in Figure 3.

Now suppose that four files (i.e., `sensor1.txt`, `sensor2.txt`, `sensor3.txt` and `sensor4.txt`) contain the sensor strain measurements:

```
Sensor 1         Sensor 2          Sensor 3         Sensor 4
-------------------------------------------------------------
   0.210           -0.225             0.125            0.110
   0.211           -0.230             0.126            0.111
   0.210           -0.232             0.127            0.110
   0.211           -0.234             0.030            0.115
   0.212           -0.236             0.100            0.120
   0.215           -0.240             0.150            0.150
   0.213           -0.240             0.130            0.145
   *****           -0.342             0.150            0.130
   *****           -0.340             0.170            ***** <- failed!
-------------------------------------------------------------
```

Things to do:

1. Download, compile, and run the DataArray.java program from the java examples web page.
2. Create four data files for the experimental data. I suggest that you call them `sensor1.txt` .. etc. Notice that sensor 4 fails, and hence contains an incomplete set of measurements.
3. Write a program (e.g., ExperimentalAnalysis.java) that will read and store each set of experimental measurements in a DataArray object. For each set of data, compute and print the maximum and minimum values, the range, average and standard deviation.

Note. Your solution should have six files: ExperimentalAnalysis.java (which you will write), DataArray.java (given), and the data files `sensor1.txt` through `sensor4.txt`.

Problem 4 (Intermediate): This problem will give you practice at using abstract classes to simplify the implementation of engineering property (e.g., position of the centroid, moments of inertia, orientation of the principal axes) computations for element cross sections. The computation of these properties can be complicated by irregular section shapes and/or cross section shapes that change as a function of loading (e.g., crack patterns in a concrete beam).

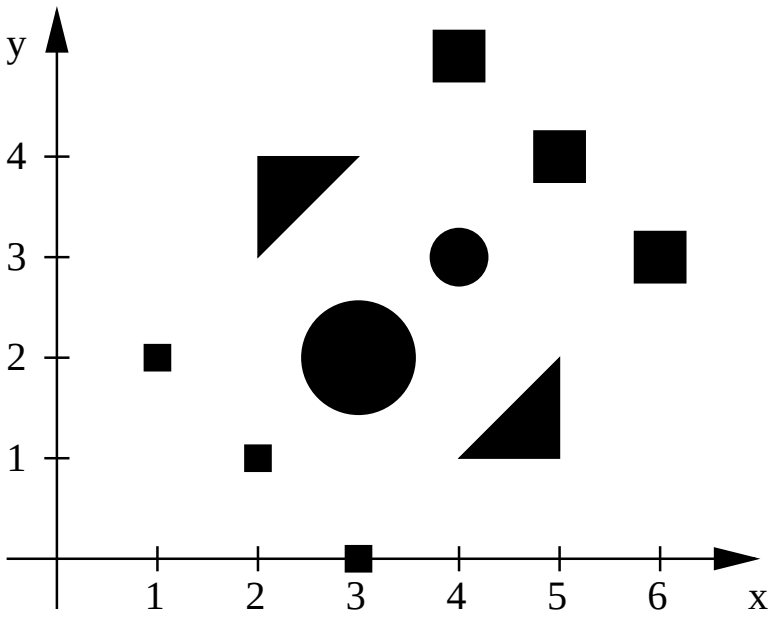

Figure 4: Spatial layout of circle, rectangle and triangle shapes.

Figure 4 shows, for example, a spatial layout of rectangle, circle and triangle shapes. The small and large rectangles have sidelength 0.25 and 0.5 respectively. The small and large circles have radius 0.5 and 1.0 respectively. Circle and rectangle shapes are positioned at their center points. Triangles are defined by the position of the three nodal/corner points.

Engineering Property Formulae. If the total number of shapes is denoted by N, then the total area of the grid, A, is given by

$$A = \sum_{i=1}^{N} A_i \tag{1}$$

The (x,y) coordinates of the grid centroid are defined by:

$$A\bar{x} = \sum_{i=1}^{N} x_i \cdot A_i \quad \text{and} \quad A\bar{y} = \sum_{i=1}^{N} y_i \cdot A_i \tag{2}$$

The area moments of inertia about the x- and y-axes are given by:

$$I_{xx} = \sum_{i=1}^{N} y_i^2 \cdot A_i \quad \text{and} \quad I_{yy} = \sum_{i=1}^{N} x_i^2 \cdot A_i \tag{3}$$

respectively. Similarly the cross moment of inertia is given by

$$I_{xy} = \sum_{i=1}^{N} x_i \cdot y_i \cdot A_i \tag{4}$$

The corresponding moments of inertia about the centroid are given by the parallel axes theorem. Finally, the orientation of the principle axes are given by

$$\tan(2\theta) = \left[\frac{2I_{xy}}{I_{xx} - I_{yy}} \right] \tag{5}$$

Things to do.

The computation of engineering properties for this spatial layout can be simplified if the basic algorithms for area, centroid, and inertia calculations are specified in terms of shapes. Java will take care of the details of calling the appropriate methods within each specific shape object.

1. Download, compile and run the abstract shape example (e.g., Shape.java, Location.java, TestShape.java) from the java examples page. Then download, compile and run the Triangle code from the java examples web page.

2. The computation of engineering properties depends on quantities such as the cross section area and centroid (i.e., (x,y) location). An algorithm will need to retrieve this information from each of the object types.

Extend the abstract shape class so that methods for retrieving the x and y coordinates are included. I suggest that you simply add the method declarations:

```
public abstract double getX();
public abstract double getY();
```

to Shape.java and then add concrete implementations of of the methods `getX()` and `getY()` to Circle.java, Rectangle.java and Triangle.java.

3. Modify the Triangle code so that it extends Shape, i.e.,

```
public class Triangle extends Shape { ....
```

Triangles are defined by the (x,y) coordinates of the three corner points. The center point of a triangle should be defined as the average of the three respective coordinate values, i.e.,

$$c.x = \left[\frac{x_1 + x_2 + x_3}{3}\right] \quad \text{and} \quad c.y = \left[\frac{y_1 + y_2 + y_3}{3}\right]. \tag{6}$$

4. Write a Java program that will initialize and position circle, rectangle and triangle shapes as shown on Figure 4, and then compute and print the grid area, x and y coordinates of the grid centroid, moments of inertia I_{xx}, I_{yy}, and I_{xy} computed about the axes/origin and, finally, the grid centroid. For details, see equations 1 through 4.

Hint. Notice that the layout of shapes in Figure 4 is symmetric about the line x = y. Hence, you should expect that: (1) the centroid will lie along this line, and (2) the principal axes will be oriented along this line.

Hint. To help you get started, here is the basic structure of my implementation for the engineering properties computation:

```
/*
 *  ================================================================
 *  InertiaComputation.java: Compute moments of inertia for a grid
 *                           of rectangular shapes.
 *
 *  Written By: Mark Austin                              November 2009
 *  ================================================================
 */
```

```
public class InertiaComputation {

   public static final int iNoShapes = 10;

   public static void main ( String args[] ) {

      // Create array of shapes

      Shape s[] = new Shape [ iNoShapes ] ;

      s[0] = new Rectangle( 0.25, 0.25, 1.0, 2.0 );

      .... details of code removed ....

      s[9] = new Rectangle( 0.5, 0.5, 6.0, 3.0 );

      // Compute and print total shape area .....

      .... details of code removed ....

      // Compute and print Ixx computed about the x-axis ....

      .... details of code removed ....

      // Compute and print Iyy computed about the y-axis ....

      .... details of code removed ....

      // Compute and print Ixy computed about the origin ....

      .... details of code removed ....

      // Compute and print the centroid coordinates origin ....

      .... details of code removed ....

      // Compute and print moments of inertia about the centroid....

      .... details of code removed ....

      // Compute and print angle of principle axes....

      .... details of code removed ....
   }
}
```

Problem 5 (Intermediate): Repeat problem 4, but this time use an array list to store the shapes.