Engineering Software Development in Java

Lecture Notes for ENCE 688R, Civil Information Systems

Spring Semester, 2016

Mark Austin,
Department of Civil and Enviromental Engineering,
University of Maryland,
College Park,
Maryland 20742, U.S.A.

8.13 Exercises

This section covers Java programming with objects. It is a work in progress!

8.1 As shown in Figure 8.12 below, rectangles may be defined by the (x,y) coordinates of corner points that are diagonally opposite.

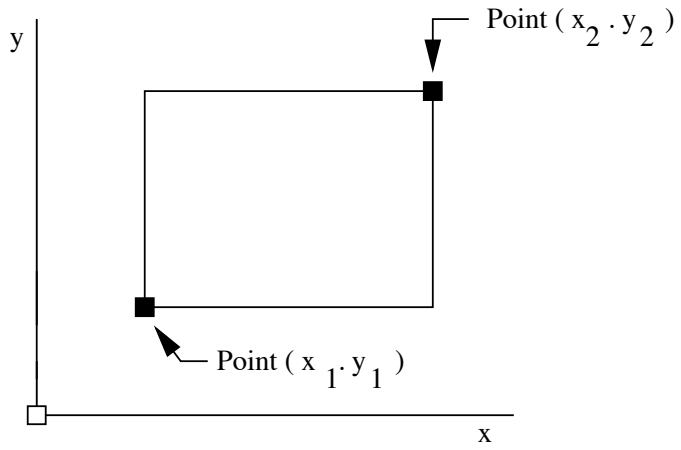

Figure 8.12. Definition of a rectangle via diagonally opposite corner points

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 2005
 *  ============================================================================
 */

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 Fri Apr 21 10:17:29 2006
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 Fri Apr 21 10:17:39 2006
```

The Rectangle class has methods to create objects (i.e, Rectangle), convert the details of a rectan-

gle 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.

8.2 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?

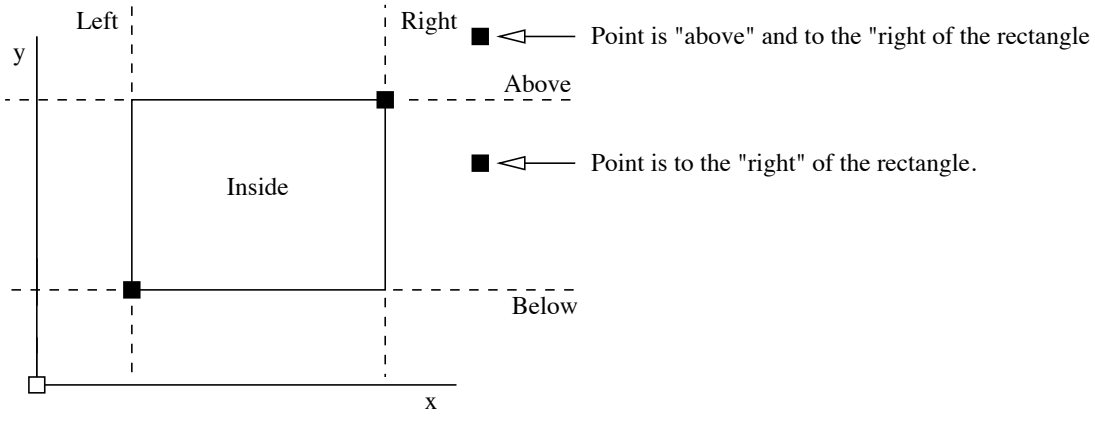

Figure 8.13. Classification of an (x,y) cooordinate relative to a rectangle

Figure 8.13 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 8.13 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.

8.3 Figure 8.14 shows three types of spatial relationship (touching, overlap, and enclosure) between two rectangle.

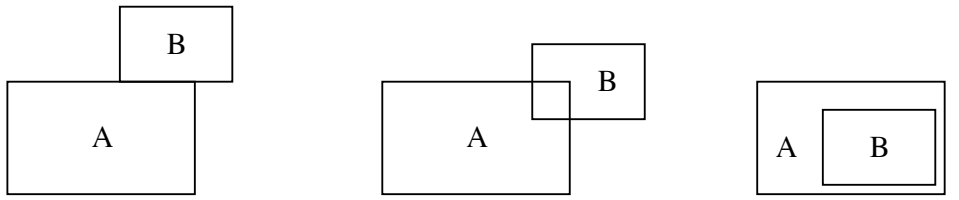

Figure 8.14. Spatial relationships between two rectangles

Extend the functionality of the Rectangle class to support the evaluation of these three types of relationships.

8.4 A data array class is a one dimensional array of floating-point numbers together with methods to compute statistics on the stored values (e.g., maximum value, minumum value). Download, compile, and run the DataArray java code from the class web site. Re-code the rainfall analysis program so that it uses the DataArray facilities.

Hint. Your solution will have two source code files, DataArray.java and RainfallAnalysis.java. Don't change any of the code in DataArray.java; just write a new version of RainfallAnalysis.java. The files before and after compilation should be:

```
                Before                          After
====================          ====================
RainfallAnalysis.java          RainfallAnalysis.java
DataArray.java                        DataArray.java
                                RainfallAnalysis.class
                                       DataArray.class
```

```
====================          ====================
```

You should find that your implementation is considerably shorter than in Problem 22.

8.5 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.

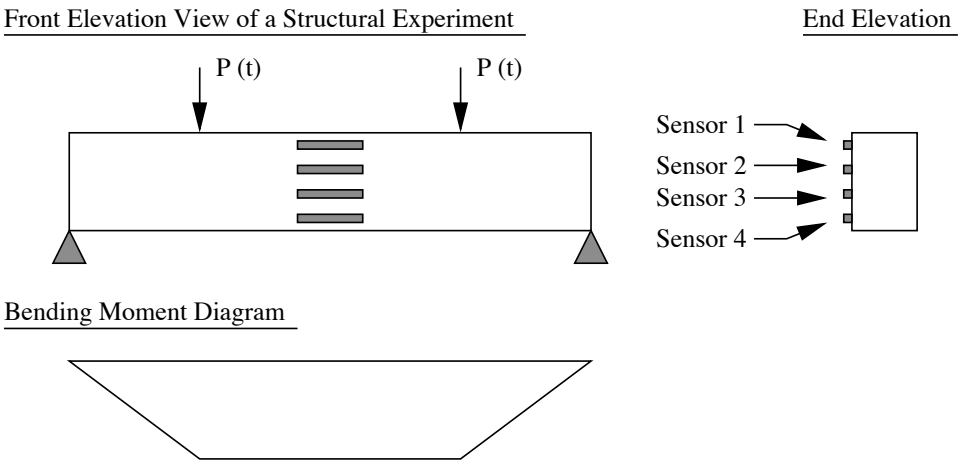

Figure 8.15. Structural experiment.

Figure 8.15 shows front and end elevation views of the experimental setup, and the bending moment diagram that will result from the symmetrically applied loading. Four sensors are attached to the center of the beam (where the bending moment will be constant. Since shear forces will be zero, any cracks that will develop will be vertical).

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
-------------------------------------------------------------
  -0.010          -0.0025            0.0025           0.010
  -0.011          -0.0030            0.0026           0.011
  -0.010          -0.0032            0.0027           0.010
  -0.011          -0.0034            0.0030           0.015
  -0.012          -0.0036            0.0100           0.120
  -0.015          -0.0040            0.0150           0.250
  -0.017          -0.0042            0.0250           ***** <- 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.

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

8.6 Suppose that we need to compute the sum of the geometric series,

$$\text{Sum}(c, n) = c + c^2 + c^3 + \cdots + c^n, \tag{8.1}$$

where "c" is a complex number of the form c = a + bi, and "n" is a positive integer. The sum of a geometric series can be computed efficiently in two ways; (1) Using Horner's rule to re-write the series as,

$$\text{Sum}(c, n) = c\,(1 + c\,(1 + \cdots c(1 + c)) \cdots)\,, \tag{8.2}$$

and (2) Using a geometric summation formula,

$$\text{Sum}(c, n) = c + c^2 + \cdots + c^n = \left[\frac{c - c^{(n+1)}}{1 - c}\right] \tag{8.3}$$

Starting with the Complex.java package handed out in class, write a Java program that will prompt a user for the coefficients a, b, and n, and then compute the series summation using Horner's rule and the geometric summation formula. In each case, show that

$$(1 + 2i) + (1 + 2i)^2 + (1 + 2i)^3 \quad \text{evaluates to} \quad -13 + 4i. \tag{8.4}$$

8.7 If a, b, and c are complex numbers, show that solutions to

$$a \cdot x^2 \; + \; b \cdot x \; + \; c = 0 \tag{8.5}$$

are given by

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{8.6}$$

Use your library of methods for complex number arithmetic and equation 8.6 to show that solutions to

$$(2.00 + 2.00i)\,x^2 \; + \; (3.00 + 3.00i)\,x \; + \; (3.00 + 5.00i) = 0.0 \tag{8.7}$$

are $x_1 = -0.5445 - 1.2164i$ and $x_2 = -0.9555 + 1.2164i$.

Check that x_1 and x_2 are in fact solutions to Equation 8.7 by evaluating each term, and arranging real and complex components in a table of output.

Hint. You need to remember that all of the arithmetic in this problem needs to work with complex numbers. Hence, suppose that you want to compute the roots to the equation:

$$2x^2 - 3x + 2 = 0.0 \tag{8.8}$$

where the coefficients are all real numbers. In this context, you need to implement the equation as if it were written:

$$(2 + 0i)x^2 - (3 + 0i)x + (2 + 0i) = 0.0 \tag{8.9}$$

Your solution should consist of two files: Complex.java and Quadratic.java. You can download Complex.java from the class web site. Quadratic.java will simply call the methods in Complex.java; therefore, there is no need to change the contents of Complex.java.

8.8 Lots of problems in Civil Engineering boil down to evaluation of some sort of geometric relationship among various kinds of objects (e.g., points, lines, areas, volumes).

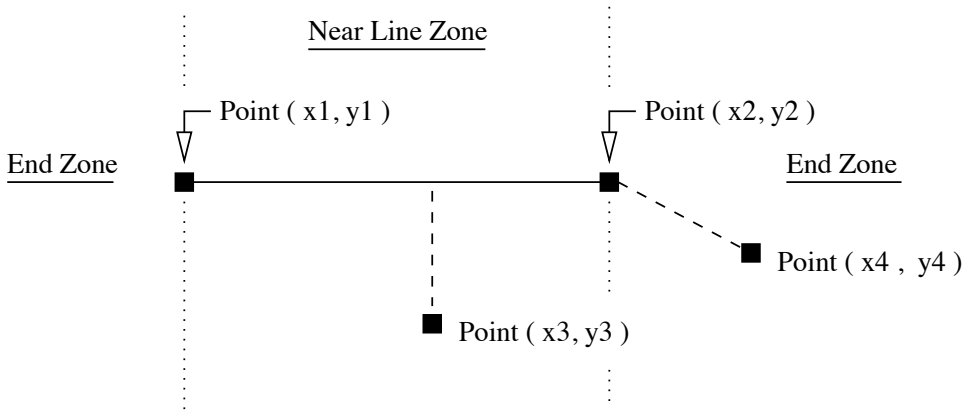

Figure 8.16. Distance of point to a line segment.

Figure 8.16 shows a line segment defined by two pairs of points (x_1, y_1) and (x_2, y_2), plus two single points located at coordinates (x_3, y_3) and (x_4, y_4).

Write a Java program that will:

1. Prompt a user for (x,y) coordinates of the line segment end points,

2. Prompt a user for (x,y) coordinates of a single point,

3. Compute and print the equation of the line segment,

4. Compute and print the distance of the single point from the line segment.

Notice that if the single point is located inside the "near line zone" then the point-to-line distance is defined by the perpendicular distance from the line to the point. This distance is defined by standard formulae. For cases where the single point lies in an "end zone" then the distance is simply the euclidean distance from the closest end point and the single point itself.