Problem 8.1: Model rectangles with Vertices ....

Source code: Here is a very short implementation of the vertex code:

```
/**
  *  ========================================================
  *  Vertex.java: Source code for two-dimensional vertex..
  *
  *  Written by: Mark Austin                      April, 2006
  *  ========================================================
  */

public class Vertex {
   protected double dX;
   protected double dY;

   // Constructor methods ....

   public Vertex() {
      dX = dY = 0.0;
   }

   public Vertex( double dX, double dY ) {
      this.dX = dX;
      this.dY = dY;
   }

   // Convert vector to a string ...

   public String toString() {
      return "Vertex(" + dX + ", " + dY + ")";
   }
}
```

Now let's move on to the rectangle code:

```
/*
 *  ============================================================
 *  Rectangle2.java : A library of methods for creating and
 *                    managing rectangles
 *
 *  Definition of Rectangles
 *  ------------------------
 *  Rectangles are defined by the (x,y) coordinates of corner
 *  points that are diagonally opposite.
 *
 *  Methods
 *  -------------------------
 *
 *  double      area() -- returns the area of a rectangle
 *  double perimeter() -- returns the perimeter of a rectangle
 *
 *  Written By: Mark Austin                            April 2006
 *  ============================================================
 */

import java.lang.Math;

public class Rectangle2 {
   protected Vertex vertex1;  // First corner point....
   protected Vertex vertex2;  // Second corner point....

   // Constructor methods ....

   public Rectangle2() {
      vertex1 = new Vertex();
      vertex2 = new Vertex();
   }
```

```
   public Rectangle2( double dX1, double dY1, double dX2, double dY2 ) {

      vertex1 = new Vertex();
      vertex1.dX = dX1;
      vertex1.dY = dY1;

      vertex2 = new Vertex();
      vertex2.dX = dX2;
      vertex2.dY = dY2;
   }

   // Convert rectangle details to a string ...

   public String toString() {
          return "Rectangle: " + vertex1.toString() + "\n" +
                 "           " + vertex2.toString() + "\n";
   }

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

   public double area() {
       return Math.abs( (vertex2.dX - vertex1.dX) *
                        (vertex2.dY - vertex1.dY) );
   }

   public double perimeter() {
       return 2.0*Math.abs( vertex2.dX - vertex1.dX ) +
              2.0*Math.abs( vertex2.dY - vertex1.dY );
   }

   // 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....

      Rectangle2 rA = new Rectangle2( 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() );
   }
}
```

Output: The program output is as follows:

```
prompt >> java Rectangle2
Rectangle test program
==========================
Rectangle: Vertex(1.0, 1.0)
           Vertex(3.0, 4.0)

Perimeter = 10.0
Area      = 6.0
prompt >>
```

Problem 8.2:

Source code: Here is a partial solution to the set operation methods:

```
/*
 *  ====================================================================
 *  Rectangle3.java : A library of methods for creating and
 *                    managing rectangles
```

```
 *
 *  Definition of Rectangles
 *  ------------------------
 *  Rectangles are defined by the (x,y) coordinates of corner points that
 *  are diagonally opposite.
 *
 *  Methods
 *  -------------------------
 *
 *  double      area() -- returns the area of a rectangle
 *  double perimeter() -- returns the perimeter of a rectangle
 *
 *  Written By: Mark Austin                                    April 2006
 *  ========================================================================
 */

import java.lang.Math;

public class Rectangle3 {
   protected Vertex vertex1;  // First corner point....
   protected Vertex vertex2;  // Second corner point....

   // Constructor methods ....

   public Rectangle3() {
      vertex1 = new Vertex();
      vertex2 = new Vertex();
   }

   public Rectangle3( double dX1, double dY1, double dX2, double dY2 ) {

      vertex1 = new Vertex();
      vertex1.dX = dX1;
      vertex1.dY = dY1;

      vertex2 = new Vertex();
      vertex2.dX = dX2;
      vertex2.dY = dY2;
   }

   // Convert rectangle details to a string ...

   public String toString() {
          return "Rectangle: " + vertex1.toString() + "\n" +
                 "           " + vertex2.toString() + "\n";
   }

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

   public double area() {
       return Math.abs( (vertex2.dX - vertex1.dX) *
                        (vertex2.dY - vertex1.dY) );
   }

   public double perimeter() {
       return 2.0*Math.abs( vertex2.dX - vertex1.dX ) +
              2.0*Math.abs( vertex2.dY - vertex1.dY );
   }


   // ====================================================
   // Evaluate (x,y) coordinate relative to a rectangle.
   // ====================================================

   public boolean isInside ( Vertex v ) {
      if ( Math.min ( vertex1.dX, vertex2.dX ) < v.dX &&
           v.dX < Math.max ( vertex1.dX, vertex2.dX ) &&
           Math.min ( vertex1.dY, vertex2.dY ) < v.dY &&
           v.dY < Math.max ( vertex1.dY, vertex2.dY ) )
         return true;
      else
```

```
         return false;
   }

   // ... details of these methods not implemented yet ....

   public boolean isOutside ( Vertex v ) {
      return false;
   }

   public boolean isOnPerimeter ( Vertex v ) {
      return false;
   }

   public boolean isAbove ( Vertex v ) {
      return false;
   }

   public boolean isBelow  ( Vertex v ) {
      return false;
   }

   public boolean isLeft ( Vertex v ) {
      return false;
   }

   public boolean isRight ( Vertex v ) {
      return false;
   }

   // ============================================
   // 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....

      Rectangle3 rA = new Rectangle3( 1.0, 1.0, 3.0, 4.0 );
      System.out.println( rA.toString() );

      // Create and print test points ....

      Vertex v1 = new Vertex( 0.0, 0.0 );
      System.out.println ( v1.toString() );
      Vertex v2 = new Vertex( 2.0, 2.0 );
      System.out.println ( v2.toString() );

      if ( rA.isInside( v1 ) == true )
           System.out.println("Vertex v1 is \"inside\" Rectangle rA");
      else
           System.out.println("Vertex v1 is not \"inside\" Rectangle rA");

      if ( rA.isInside( v2 ) == true )
           System.out.println("Vertex v2 is \"inside\" Rectangle rA");
      else
           System.out.println("Vertex v2 is not \"inside\" Rectangle rA");

   }
}
```

Output: The program output is as follows:

```
prompt >> java Rectangle3
Rectangle test program
==========================
Rectangle: Vertex(1.0, 1.0)
           Vertex(3.0, 4.0)

Vertex(0.0, 0.0)
Vertex(2.0, 2.0)
```

```
Vertex v1 is not "inside" Rectangle rA
Vertex v2 is "inside" Rectangle rA
prompt >>

------------------------------------------------------------------------
Problem 8.6: Comparing the sum of geometric series composed of complex
numbers.

Source code: Here we assume that the files:

    Complex.java
    GeometricSeries.java

are in the same folder. The implementation uses getTextFromConsole()
to read lines of input provided at the keyboard. You could also use
the class Scanner.

/*
 *  ====================================================================
 *  GeometricSeries.java : Compute sum of a geometric series of complex
 *                         objects.
 *
 *  Written By : Mark Austin                                 March 2017
 *  ====================================================================
 */

import java.lang.Math;
import java.util.*;
import java.io.*;

public class GeometricSeries {

   public static void main( String args[] ) {
   Complex cA = new Complex();
   int    NoTermsInSeries;
   String sLine;

      // Print Welcome Message

      System.out.println("Welcome to the Geometriic Series Computer");
      System.out.println("-----------------------------------------");

      // Prompt user for coefficients in geometric series...

      System.out.println("Please enter complex number \"a\"");

      // Coefficient cA ....

      System.out.print("Coefficent a : Real : ");
      sLine    = getTextFromConsole();
      cA.dReal = Double.valueOf(sLine).doubleValue();

      System.out.print("Coefficent a : Imaginary : ");
      sLine         = getTextFromConsole();
      cA.dImaginary = Double.valueOf(sLine).doubleValue();

      // Number of terms in the series .....

      System.out.println("Please enter number of terms in series \"a\"");

      System.out.print("No of terms in series : ");
      sLine = getTextFromConsole();
      NoTermsInSeries = Integer.valueOf(sLine).intValue();

      // Print details of input to screen ...

      System.out.print("The complex no you have entered is :");
      System.out.println("s = " + cA.toString() );
      System.out.println("No of terms in series is = " + NoTermsInSeries );

      // Check that NoTermsInSeries is greater than or equal to one..

      if ( NoTermsInSeries < 1 ) {
```

```
      System.out.println("ERROS: No of terms in series must be at least one:");
      return;
   }

   // Use basic for loop to compute sumation of geometric series .....

   System.out.println( "Summation with basic for-loop        " );
   System.out.println( "======================================= " );

   Complex cSum = new Complex();
   for (int ii = 1; ii <= NoTermsInSeries; ii = ii + 1) {

       // Compute s^(ii) ......

       Complex cB   = new Complex();
       for (int ij = 1; ij <= ii; ij = ij + 1) {
            if (ij == 1)
                cB = cB.Add( cA );
            else
                cB = cB.Mult( cA );
       }

       // Add s^(ii) to series sum ......

       cSum = cSum.Add( cB );
   }
   System.out.println( "Summation 1 = " + cSum.toString() );

   // Compute sum of geometric series using Horner's rule ....

   System.out.println( "" );
   System.out.println( "Summation using Horner's formula" );
   System.out.println( "======================================= " );

   Complex cI = new Complex();
   cI.dReal = 1.0; cI.dImaginary = 0.0;

   Complex cSum2 = new Complex();
   cSum2.dReal      = cA.dReal;
   cSum2.dImaginary = cA.dImaginary;

   for (int ij = 1; ij <= NoTermsInSeries - 1; ij = ij + 1) {
        cSum2 = cA.Mult( cSum2.Add(cI) );
   }

   System.out.println( "Summation 2 = " + cSum2.toString() );

   // Use geometric formula to series sum....

   System.out.println( "" );
   System.out.println( "Summation with geometric-series formula " );
   System.out.println( "======================================= " );

   Complex cB = new Complex();
   for (int ij = 1; ij <= NoTermsInSeries; ij = ij + 1) {
            if (ij == 1)
                cB = cB.Add( cA );
            else
                cB = cB.Mult( cA );
   }

   Complex cNumerator   = cA.Mult(cI.Sub(cB));
   Complex cDenominator = cI.Sub(cA);
   Complex cSum3 = cNumerator.Div( cDenominator );
   System.out.println( "Summation 3 = " + cSum3.toString() );

}

/*
 *  ============================================================
 *  Method getTextFromConsole(): Read line of text from console
 *                               (keyboard input)....
 *
```

```
    *  Input  : None.
    *  Output : String inLine -- character string of keyboard input
    *  ============================================================
    */

    public static String getTextFromConsole() {
       String inLine = "";

       // Create buffered reader for keyboard input stream....

       BufferedReader inStream = new BufferedReader (
                                 new InputStreamReader(System.in));

       // Try to read input from keyboard ....

       try {
          inLine = inStream.readLine();
       } catch (IOException e) {
          System.out.println("IOException: " + e);
       }

       return inLine;
    }
}
```

Output:

```
Script started on Sun Mar 26 19:38:05 2017
prompt >> java GeometricSeries
Welcome to the Geometriic Series Computer
-----------------------------------------
Please enter complex number "a"
Coefficent a : Real : 1
Coefficent a : Imaginary : 2
Please enter number of terms in series "a"
No of terms in series : 3
The complex no you have entered is :s = 1.0+2.0i
No of terms in series is = 3
Summation with basic for-loop
=====================================
Summation 1 = -13.0+4.0i

Summation using Horner's formula
=====================================
Summation 2 = -13.0+4.0i

Summation with geometric-series formula
=====================================
Summation 3 = -13.0+4.0i
prompt >> exit
Script done on Sun Mar 26 19:38:26 2017
```

Problem 8.8: Compute the distance between a point and a line segment.

My solution makes use of three classes: Node.java, Vector2D.java (for the arithmetic), and LineSegment.java (to setup and exercise the points and line segment).

The details of Node.java are as follows:

```
/**
  *  ==========================================================
  *  Node.java: Java class for nodes in a simple polygon.
  *             The class node extends class vector.
  *
  *  Written by: Mark Austin                    November, 2004
  *  ==========================================================
  */

public class Node extends Vector2D {
   protected String sName;
```

```
   // Constructor methods ....

   public Node() {
      super( 0.0, 0.0 );
   }

   public Node( double dX, double dY ) {
      super( dX, dY );
   }

   public Node( String sName, double dX, double dY ) {
      super( dX, dY );
      this.sName = sName;
   }

   // Set name for the node ...

   public void setName( String sName ) {
      this.sName = sName;
   }

   // Retrieve x and y coordinates ...

   public double getX() { return dX; }
   public double getY() { return dY; }

   // Convert node to a string ...

   public String toString() {
      return "Node(\"" + sName + "\") is at (" + dX + "," + dY + ")";
   }

   // Exercise methods in the Node class .....

   public static void main( String args[] ) {

      // Create and print "point 1", a node at coordinate (1,2)...

      Node nA = new Node();
      nA.dX = 1.0;
      nA.dY = 2.0;
      nA.sName  = "Point 1";

      System.out.println( nA.toString() );

      // Create and print "point 2", a node at coordinate (5,2)...

      Node nB = new Node ("Point 2", 5.0, 5.0 );
      System.out.println( "\n" + nB );
      System.out.println( "Vector Magnitude =" + nB.length() );

   }
}
```

The details of Vector2D.java are as follows:

```
/**
  *  ======================================================
  *  Vector2D.java: Source code for two-dimensional vectors
  *
  *  Written by: Mark Austin                  November, 2005
  *  ======================================================
  */

import java.lang.Math;

public class Vector2D {

   protected double dX;
   protected double dY;

   // Constructor methods ....
```

```
public Vector2D() {
   dX = dY = 0.0;
}

public Vector2D( double dX, double dY ) {
   this.dX = dX;
   this.dY = dY;
}

// Convert vector to a string ...

public String toString() {
   return "Vector2D(" + dX + ", " + dY + ")";
}

// Compute magnitude of vector ....

public double length() {
   return Math.sqrt ( dX*dX + dY*dY );
}

// Sum of two vectors ....

public Vector2D add( Vector2D v1 ) {
    Vector2D v2 = new Vector2D( this.dX + v1.dX, this.dY + v1.dY );
    return v2;
}

// Subtract vector v1 from v .....

public Vector2D sub( Vector2D v1 ) {
    Vector2D v2 = new Vector2D( this.dX - v1.dX, this.dY - v1.dY );
    return v2;
}

// Scale vector by a constant ...

public Vector2D scale( double scaleFactor ) {
    Vector2D v2 = new Vector2D( this.dX*scaleFactor, this.dY*scaleFactor );
    return v2;
}

// Normalize a vectors length....

public Vector2D normalize() {
   Vector2D v2 = new Vector2D();

   double length = Math.sqrt( this.dX*this.dX + this.dY*this.dY );
   if (length != 0) {
     v2.dX = this.dX/length;
     v2.dY = this.dY/length;
   }

   return v2;
}

// Dot product of two vectors .....

public double dotProduct ( Vector2D v1 ) {
     return this.dX*v1.dX + this.dY*v1.dY;
}

// Exercise methods in Vector2D class

public static void main ( String args[] ) {
   Vector2D vA = new Vector2D( 1.0, 2.0);
   Vector2D vB = new Vector2D( 2.0, 2.0);

   System.out.println( "Vector vA =" + vA.toString() );
   System.out.println( "Vector vB =" + vB.toString() );

   System.out.println( "Vector vA-vB =" + vA.sub(vB).toString() );
   System.out.println( "Vector vB-vA =" + vB.sub(vA).toString() );
```

```
      System.out.println( "vA.normalize() =" + vA.normalize().toString() );
      System.out.println( "vB.normalize() =" + vB.normalize().toString() );

      System.out.println( "Dot product vA.vB =" + vA.dotProduct(vB) );
      System.out.println( "Dot product vB.vA =" + vB.dotProduct(vA) );
   }
}
```

And now, the details of LineSegment.java are:

```
/*
 *  ========================================================================
 *  LineSegment.java: A line segment is defined by the (x,y) coordinates of
 *              its two end points.
 *
 *  Written By: Mark Austin                                    November 2005
 *  ========================================================================
 */

import java.lang.Math;

public class LineSegment {
   protected Node n1, n2; // nodal points defining the LineSegment

   // Constructor method : just create instance of class.

   public LineSegment() { }

   // Create instance of class and set (x,y) coordinates.

   public LineSegment( double dX1, double dY1,
                       double dX2, double dY2 ) {

      setLineSegment( dX1, dY1, dX2, dY2 );
   }

   // Set x and y coordinates of LineSegment.

   public void setLineSegment( double dX1, double dY1,
                               double dX2, double dY2 ) {
      n1 = new Node( dX1, dY1 );
      n2 = new Node( dX2, dY2 );
   }

   // Get end points 1 and 2.

   public Node getNode1() { return n1; }
   public Node getNode2() { return n2; }

   // Print details of line segment.

   public void printSegment() {

      System.out.println("Line Segment");
      System.out.println("Node 1 : (x,y) = " + n1.toString() );
      System.out.println("Node 2 : (x,y) = " + n2.toString() );
   }

   // Compute length of line segment.

   public double segmentLength() {
      double dLength;

      dLength = (n1.getX() - n2.getX())*(n1.getX() - n2.getX()) +
                (n1.getY() - n2.getY())*(n1.getY() - n2.getY());

      return ((double) Math.sqrt(dLength));
   }

   // Compute min distance from line segment to point ...

   public double distanceToPoint( Node pt1 ) {
```

```
      // Get vector direction along line segment ...

      Vector2D direction = n2.sub(n1);

      // Compute intercept point ....

      double rhs = pt1.sub(n1).dotProduct ( direction );
      double lhs =   direction.dotProduct ( direction );
      double t = rhs/lhs;

      // If intercept lies in "end zones" compute Euclidean distance
      // to end points of line segments....

      if ( 0.0 < t && t < 1.0 )
           return pt1.sub( n1.add( direction.scale(t)) ).length();
      else
           return Math.min ( pt1.sub( n1 ).length(), pt1.sub( n2 ).length() );
   }

   // ----------------------------------------
   // Exercise methods in line segment class.
   // ----------------------------------------

   public static void main( String args[] ) {
   double dX, dY;

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

      // Create two new line segments.

      LineSegment s1 = new LineSegment();
      s1.setLineSegment( 1.0, 1.0, 4.0, 4.0 );

      LineSegment s2 = new LineSegment( 1.5, 1.5, 1.5, 4.5 );
      s2.n1.setName( "n1" );
      s2.n2.setName( "n2" );

      // Print details of line segments.

      s1.printSegment();
      s2.printSegment();

      // Compute length of line segments.

      System.out.println("Segment1 has length : " + s1.segmentLength());
      System.out.println("Segment2 has length : " + s2.segmentLength());

      // Compute distance of a point from a length segment....

      Node pt1 = new Node ( "pt1", 0.0,  0.0 );
      System.out.println( pt1.toString() + ": Distance to line = " +
                          s1.distanceToPoint( pt1 ) );
      Node pt2 = new Node ( "pt2", 1.0,  0.0 );
      System.out.println( pt2.toString() + ": Distance to line = " +
                          s1.distanceToPoint( pt2 ) );
      Node pt3 = new Node ( "pt3", 5.0,  1.0 );
      System.out.println( pt3.toString() + ": Distance to line = " +
                          s1.distanceToPoint( pt3 ) );
      Node pt4 = new Node ( "pt4", 6.0, -1.0 );
      System.out.println( pt4.toString() + ": Distance to line = " +
                          s1.distanceToPoint( pt4 ) );

      // End of exercise.

      System.out.println("==============================");
      System.out.println("End of LineSegment test program");
   }
}
```

A script of the program input/output is:

```
Script started on Mon Mar 27 10:44:49 2017
prompt >>
prompt >> java LineSegment
LineSegment test program
==============================
Line Segment
Node 1 : (x,y) = Node("null") is at (1.0,1.0)
Node 2 : (x,y) = Node("null") is at (4.0,4.0)
Line Segment
Node 1 : (x,y) = Node("n1") is at (1.5,1.5)
Node 2 : (x,y) = Node("n2") is at (1.5,4.5)
Segment1 has length : 4.242640687119285
Segment2 has length : 3.0
Node("pt1") is at (0.0,0.0): Distance to line = 1.4142135623730951
Node("pt2") is at (1.0,0.0): Distance to line = 1.0
Node("pt3") is at (5.0,1.0): Distance to line = 2.8284271247461903
Node("pt4") is at (6.0,-1.0): Distance to line = 4.949747468305833
==============================
End of LineSegment test program
prompt >>
prompt >> exit
Script done on Mon Mar 27 10:45:08 2017
```
