Department of Civil and Environmental Engineering, Spring Semester, 2023

ENCE 688R: Midterm Exam: 24 Hours, Open Book and Open Notes

Name : _____

Exam Format, Submission, Grading. This take home midterm exam is open notes and open book. You need to comply with the university regulations for academic integrity.

There are two questions. Partial credit will be given for partially correct answers, so please show all your working. See the **class web page for instructions on how to submit your exam paper**.

Question	Points	Score
1	50	
2	50	
Total	100	

Question 1: 50 points. This question covers **Programming with Python** and is concerned with computation of the shortest distance between a point and a line segment.

Problem Setup. Figure 1 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). The dotted lines show the shortest distance between the single points and the line segment.

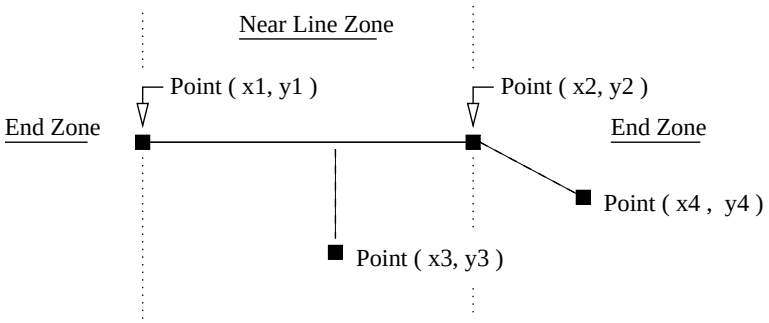

Figure 1: Dashed lines represent the shortest distance of point to a line segment.

From a computational standpoint, two cases exist: (1) the point is in the "near line zone," and (2) the point is in the "end zone." Solutions to case 1 require computation of the length of a line that is perpendicular to the line segment and passes through the point. The math for this involves vector arithmetic (e.g., dot products) – don't panic, I will only ask questions about details of the software implementation. Solutions to case 2 simply require use of the Pythagorean theorem (i.e., $a^2 + b^2 = c^2$).

Source Code and Test Program. The source code for solutions to this problem is organized into three classes (i.e., Vector2D, Node2D and LineSegment2D) and two test progams (i.e., TestVector2D and TestLineSegment).

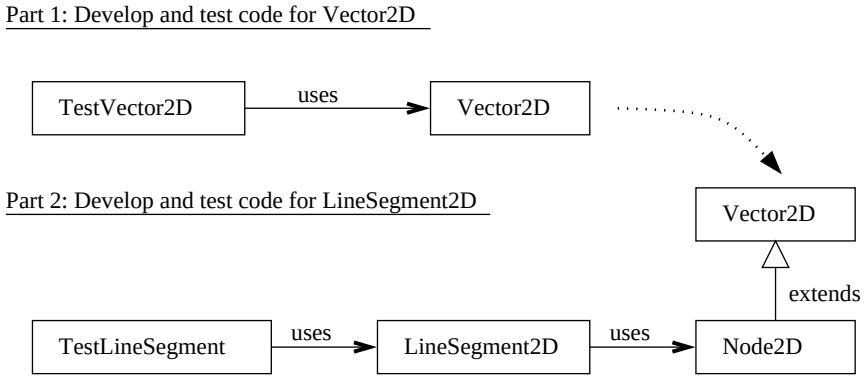

Figure 2: Schematic of test programs and arrangement of classes.

First, we develop and test Vector2D (see the upper section of Figure 2). The second test program

exercises the Node2D and LineSegment2D classes by assembling the arrangement of line segments (i.e, see $s1$ and $s2$) and data points (i.e, see $pt1$, $pt2$, $pt3$ and $pt4$) shown in Figure 3.

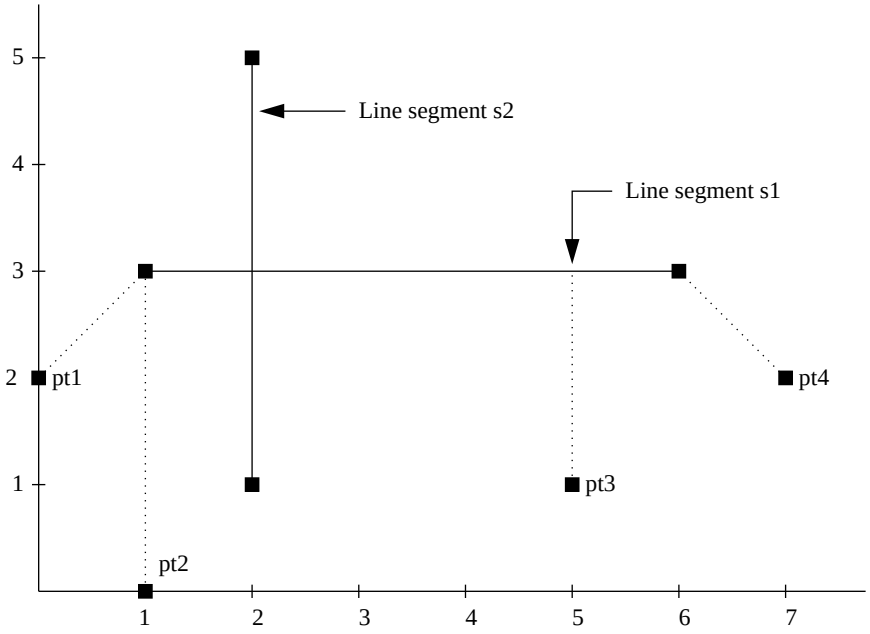

Figure 3: Two-dimensional coordinate system containing line segements and points.

Notice that points $pt1$ and $pt4$ are in the end zone; points $pt2$ and $pt3$ are in the "near line" zone.

Details of the Python source code files are as follows:

File 1: Vector2D.py

source code

```
# ====================================================================
# Vector2D.py: Model two-dimensional vectors and vector and arithmetic.
#
# Written by: Mark Austin                                    April 2023
# ====================================================================

import math

class Vector2D:

   # Constructor method ...

   def __init__(self, dX, dY):
       self._dX = dX
       self._dY = dY
```

```
# Set/get name ...

def setName(self, name ):
    self._name = name

def getName(self):
   return self._name

# Set/get x and y coordinates ...

def setX(self, dX ):
    self._dX = dX

def getX(self):
   return self._dX

def setY(self, dY ):
   self._dY = dY

def getY(self):
   return self._dY

# get current position

def getPosition(self):
   return self._dX, self._dY

# Scale vector by a constant ...

def scale( self, scaleFactor ):
   return Vector2D( self._dX*scaleFactor, self._dY*scaleFactor )

# Normalize a vectors length to one ...

def normalize(self):
   length = self.length()
   return Vector2D( self._dX/length, self._dY/length )

# Compute length of vector vector .....

def length(self):
   return math.sqrt( self._dX*self._dX + self._dY*self._dY )

# Compute dot product of two vectors .....

def dotProduct ( self, v1 ):
   return self._dX*v1.getX() + self._dY*v1.getY()

# Overload + operator

def __add__(self, vector2 ):
   return Vector2D(self._dX + vector2.getX(), self._dY + vector2.getY() )

# Overload - operator
```

```
   def __sub__(self, vector2 ):
      return Vector2D(self._dX - vector2.getX(), self._dY - vector2.getY() )

   # Return string represention of vector object ...

   def __str__(self):
       return "Vector2D ( {:6.2f}, {:6.2f} )".format( self._dX, self._dY )
```

File 2: Node2D.py

source code

```
# ========================================================
# Node2D.py: Simplified model of nodes in two domensions.
#
# Modified by: Mark Austin                   October, 2020
# ========================================================

import math

from Vector2D import Vector2D

class Node2D(Vector2D):

    def __init__(self, dX, dY):
        Vector2D.__init__(self, dX, dY)

    # Move x & y coordinates by p & q

    def move(self, p, q):
        self.__dY += p
        self.__dY += q

    # Compute distance between two points ...

    def distance(self, second):
        x_d = self._dX - second._dX
        y_d = self._dY - second._dY
        return (x_d**2 + y_d**2)**0.5

    # Return string represention of object ...

    def __str__(self):
        return "Node2D:( {:5.2f}, {:5.2f} )".format( self._dX, self._dY )
```

File 3: LineSegment2D.py

source code

```
# ====================================================================
# LineSegment2D.py: Line segments are defined by end points (x1, y1)
#                   and (x2, y2).
#
# Written by: Mark Austin                                  April 2023
# ====================================================================

import math

from Node2D import Node2D

class LineSegment2D:
  __length = 0
  __angle  = 0

  # Line segment constructor methods ...

  def __init__(self, x1, y1, x2, y2 ):
     self.__pt1 = Node2D(x1,y1)
     self.__pt2 = Node2D(x2,y2)
     self.__length = self.__pt1.distance(self.__pt2)
     self.__angle  = self.getAngle()

  # Get end points pt1 and pt2 ...

  def getNode1(self):
     return self.__pt1;

  def getNode2(self):
     return self.__pt2;

  # Set/get line segment name ...

  def setName( self, name ):
     self.name = name;

  # Compute length of line segment.

  def getLength(self):
     dX = self.__pt1.getX() - self.__pt2.getX()
     dY = self.__pt1.getY() - self.__pt2.getY()

     return math.sqrt(dX*dX + dY*dY)

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

  def distanceToPoint( self, pt1 ):
      n1 = self.getNode1()
      n2 = self.getNode2()

      # Get vector direction along line segment ...

      direction = n2 - n1
```

```
    # Compute intercept point ....

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

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

    if 0.0 < t and t < 1.0:
       distance = (pt1 - (n1 + direction.scale(t))).length();
    else:
       distance = min ( (pt1 - n1).length(), (pt1-n2).length() )

    return distance;

# Compute angle (radians) for coordinates in four quadrants ....

def getAngle(self):
   dX = self.__pt2.getX() - self.__pt1.getX();
   dY = self.__pt2.getY() - self.__pt1.getY();

   if dY >  0.0 and dX == 0.0:
      angle = math.pi/2.0
   if dY >= 0.0 and dX > 0.0:
      angle = math.atan( dY/dX )
   if dY >= 0.0 and dX < 0.0:
      angle = math.pi + math.atan( dY/dX )
   if dY < 0.0 and dX < 0.0:
      angle = math.pi + math.atan( dY/dX )
   if dY < 0.0 and dX >= 0.0:
      angle = 2*math.pi + math.atan( dY/dX )

   return angle

# String represention of line segment ...

def __str__(self):
   x1 = self.__pt1.getX();
   y1 = self.__pt1.getY();
   x2 = self.__pt2.getX();
   y2 = self.__pt2.getY();
   return "--- LineSegment2D: (x1,y1) = ({:5.2f}, {:5.2f}), (x2,y2) = ({:5.2f}, {:5.2f})".format( x1, y1, x2, y2
```

Test Program for Vector2D. The test program for Vector2D.py, i.e.,

source code

```
# ====================================================
# TestVector2D.py: Exercise Vector2D class ...
# ====================================================
```

```
from Vector2D import Vector2D

# main method ...

def main():
    print("--- Enter TestVector2D.main()       ... ");
    print("--- ============================== ... ");

    print("--- Part 1: Create test vectors ... ");

    v1 = Vector2D(1.0, 2.0)
    v2 = Vector2D(2.0, 4.0)
    v3 = Vector2D(3.0, 4.0)

    print(v1)
    print(v2)
    print(v3)

    print("--- Part 2: Compute length of vectors ... ");

    print("---   v1.length() --> {:6.2f} ...".format( v1.length() ))
    print("---   v2.length() --> {:6.2f} ...".format( v2.length() ))
    print("---   v3.length() --> {:6.2f} ...".format( v3.length() ))

    print("--- Part 3: Compute dot products ... ");

    print("---   dot product (v1,v2) = {:6.2f} ...".format( v1.dotProduct(v2) ))
    print("---   dot product (v1,v3) = {:6.2f} ...".format( v1.dotProduct(v3) ))

    print("--- Part 4: Vector scale ... ");

    print("---   v1.scale(2.0) --> {:s} ...".format( v1.scale(2.0).__str__() ))
    print("---   v2.scale(3.0) --> {:s} ...".format( v2.scale(3.0).__str__() ))
    print("---   v3.scale(4.0) --> {:s} ...".format( v3.scale(4.0).__str__() ))

    print("--- Part 5: Normalize vectors ... ");

    print("---   v1.normalize() --> {:s} ...".format( v1.normalize().__str__() ))
    print("---   v2.normalize() --> {:s} ...".format( v2.normalize().__str__() ))
    print("---   v3.normalize() --> {:s} ...".format( v3.normalize().__str__() ))

    print("--- Part 6: Compute vector arithmetic ... ");

    v4 = v1 + v2
    print("---   arithmetic v1 + v2 --> {:s} ...".format( v4.__str__() ))
    print("---   arithmetic v1 + v3 --> {:s} ...".format( (v1+v3).__str__() ))
    print("---   arithmetic v1 - v2 --> {:s} ...".format( (v1-v2).__str__() ))
    print("---   arithmetic v1 - v3 --> {:s} ...".format( (v1-v3).__str__() ))

    print("--- ============================== ... ");
    print("--- Finished TestVector2D.main()    ... ");

# call the main method ...
```

```
main()
```

generates the output:

```
--- Enter TestVector2D.main()       ...
--- ============================== ...
--- Part 1: Create test vectors ...
Vector2D (   1.00,   2.00 )
Vector2D (   2.00,   4.00 )
Vector2D (   3.00,   4.00 )
--- Part 2: Compute length of vectors ...
---   v1.length() -->   2.24 ...
---   v2.length() -->   4.47 ...
---   v3.length() -->   5.00 ...
--- Part 3: Compute dot products ...
---   dot product (v1,v2) =  10.00 ...
---   dot product (v1,v3) =  11.00 ...
--- Part 4: Vector scale ...
---   v1.scale(2.0) --> Vector2D (   2.00,   4.00 ) ...
---   v2.scale(3.0) --> Vector2D (   6.00,  12.00 ) ...
---   v3.scale(4.0) --> Vector2D (  12.00,  16.00 ) ...
--- Part 5: Normalize vectors ...
---   v1.normalize() --> Vector2D (   0.45,   0.89 ) ...
---   v2.normalize() --> Vector2D (   0.45,   0.89 ) ...
---   v3.normalize() --> Vector2D (   0.60,   0.80 ) ...
--- Part 6: Compute vector arithmetic ...
---   arithmetic v1 + v2 --> Vector2D (   3.00,   6.00 ) ...
---   arithmetic v1 + v3 --> Vector2D (   4.00,   6.00 ) ...
---   arithmetic v1 - v2 --> Vector2D (  -1.00,  -2.00 ) ...
---   arithmetic v1 - v3 --> Vector2D (  -2.00,  -2.00 ) ...
--- ============================== ...
--- Finished TestVector2D.main()    ...
```

Test Program for LineSegment2D. The test program for LineSegment2D.py is as follows:

source code

```
# ====================================================
# TestLineSegment.py: Exercise line segment class ...
# ====================================================

from LineSegment2D import LineSegment2D
from Node2D import Node2D

# main method ...

def main():
    print("--- Enter TestLineSegment.main()                    ... ");
    print("--- ============================================== ... ");
```

```
    print("--- Part 1: Create line segment s1 ... ");

    s1 = LineSegment2D( 1.0, 3.0,  6.0,  3.0 )
    s1.getNode1().setName("n1");
    s1.getNode2().setName("n2");
    s1.setName("sA");

    print(s1)

    print("--- Part 2: Create line segment s2 ... ");

    n1 = Node2D( 2.0, 1.0 );
    n1.setName( "n3" );
    n2 = Node2D( 2.0, 5.0 );
    n2.setName( "n4" );

    s2 = LineSegment2D( n1.getX(), n1.getY(), n2.getX(), n2.getY() )
    s2.setName("sB");

    print(s2)

    print("--- Part 3: Compute distance of a point from a length segment ...")

    pt1 = Node2D( 0.0, 2.0 );
    distance = s1.distanceToPoint( pt1 )
    print("--- Shortest distance from s1 to {:s} --> {:5.3f}... ".format( pt1.__str__(), distance) );

    pt1 = Node2D( 1.0, 0.0 );
    distance = s1.distanceToPoint( pt1 )
    print("--- Shortest distance from s1 to {:s} --> {:5.3f}... ".format( pt1.__str__(), distance) );

    pt1 = Node2D( 5.0, 1.0 );
    distance = s1.distanceToPoint( pt1 )
    print("--- Shortest distance from s1 to {:s} --> {:5.3f}... ".format( pt1.__str__(), distance) );

    pt1 = Node2D( 7.0, 2.0 );
    distance = s1.distanceToPoint( pt1 )
    print("--- Shortest distance from s1 to {:s} --> {:5.3f}... ".format( pt1.__str__(), distance) );

    print("--- ============================================== ... ");
    print("--- Finished TestLineSegment.main()                ... ");

# call the main method ...

main()
```

---

generates the output:

```
--- Enter TestLineSegment.main()                    ...
--- ============================================== ...
```

```
--- Part 1: Create line segment s1 ...
--- LineSegment2D: (x1,y1) = ( 1.00,  3.00), (x2,y2) = ( 6.00,  3.00)
--- Part 2: Create line segment s2 ...
--- LineSegment2D: (x1,y1) = ( 2.00,  1.00), (x2,y2) = ( 2.00,  5.00)
--- Part 3: Compute distance of a point from a length segment ...
--- Shortest distance from s1 to Node2D:(  0.00,  2.00 ) --> 1.414...
--- Shortest distance from s1 to Node2D:(  1.00,  0.00 ) --> 3.000...
--- Shortest distance from s1 to Node2D:(  5.00,  1.00 ) --> 2.000...
--- Shortest distance from s1 to Node2D:(  7.00,  2.00 ) --> 1.414...
--- ============================================== ...
--- Finished TestLineSegment.main()                ...
```

Please look at the source code carefully, and then answer the following questions:

PART 1: Develop and test code for Vector2D.py.

[**1a**] (5 pts). What does the line:

```
import math
```

do in the Vector2D source code? And why is it included in this program?

[**1b**] (5 pts). Briefly explain the purpose of the lines:

```
def __init__(self, dX, dY):
    self._dX = dX
    self._dY = dY
```

[**1c**] (5 pts). The declaration for variables dX and dY in Vector2D is:

```
self._dX = dX
self._dY = dY
```

What constraints on access to dX and dY are imposed by use of the underscore?

[**1d**] (5 pts). Briefly explain the purpose of the method:

```
def __add__(self, vector2 ):
   return Vector2D(self._dX + vector2.getX(), self._dY + vector2.getY() )
```

[**1e**] (5 pts). The pair of lines:

```
def __str__(self):
    return "Vector2D ( {:6.2f}, {:6.2f} )".format( self._dX, self._dY )
```

generates a string representation for Vector2D objects. Briefly explain how the formatting specifications work.

PART 2: Extending Vector2D to create Node2D.

[**1f**] (5 pts). Briefly explain the purpose of the block of code:

```
class Node2D(Vector2D):

    def __init__(self, dX, dY):
        Vector2D.__init__(self, dX, dY)
```

[**1g**] (5 pts). Suppose that the declaration for variables dX and dY in Vector2D.py is changed from `_dX` and `_dY` to `__dX` and `__dY`. How would this affect the block of code:

```
    def __str__(self):
        return "Node2D:( {:5.2f}, {:5.2f} )".format( self._dX, self._dY )
```

at the bottom of Node2D.py?

PART 3: Develop and test code for LineSegment2D.

[**1h**] (10 pts). The block of code:

```
def __init__(self, x1, y1, x2, y2 ):
   self.__pt1 = Node2D(x1,y1)
   self.__pt2 = Node2D(x2,y2)
   self.__length = self.__pt1.distance(self.__pt2)
   self.__angle  = self.getAngle()
```

creates line segment objects. Draw and label a diagram that shows the organizational arrangement of memory that results from the method call:

```
s1 = LineSegment2D( 1.0, 3.0,  6.0,  3.0 )
s1.getNode1().setName("n1");
s1.getNode2().setName("n2");
s1.setName("sA");
```

All reasonable answers will be accepted.

[**1i**] (5 pts). The point-to-line-segment distances are computed and printed with the block of statements:

```
pt1 = Node2D( 0.0, 2.0 );
distance = s1.distanceToPoint( pt1 )
print("--- Shortest distance from s1 to {:s} --> {:5.3f}... ".format( pt1.__str__(), distance) );
```

Briefly explain how string argument to print() is systematically assembled.

Question 2: 50 points This question covers basic **Programming with Java** and is concerned with computation of the shortest distance between a point and a line segment.

Problem Setup. Figure 4 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). The dotted lines show the shortest distance between the single points and the line segment.

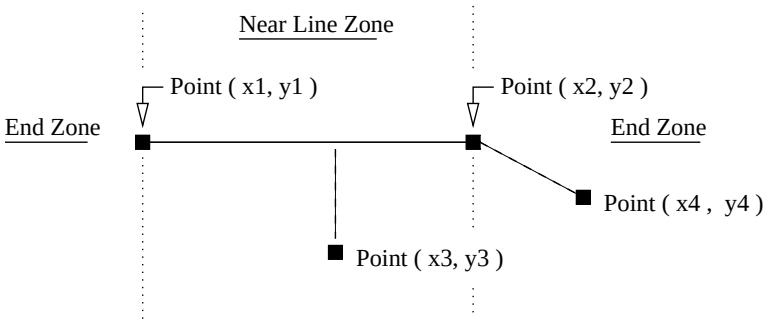

Figure 4: Dashed lines represent the shortest distance of point to a line segment.

From a computational standpoint, two cases exist: (1) the point is in the "near line zone," and (2) the point is in the "end zone." Solutions to case 1 require computation of the length of a line that is perpendicular to the line segment and passes through the point. The math for this involves vector arithmetic (e.g., dot products) – don't panic, I will only ask questions about details of the software implementation. Solutions to case 2 simply require use of the Pythagorean theorem (i.e., $a^2 + b^2 = c^2$).

Source Code and Test Program. The source code for solutions to this problem is organized into three classes (i.e., Vector2D, Node2D and LineSegment2D) and two test progams (i.e., TestVector2D and TestLineSegment).

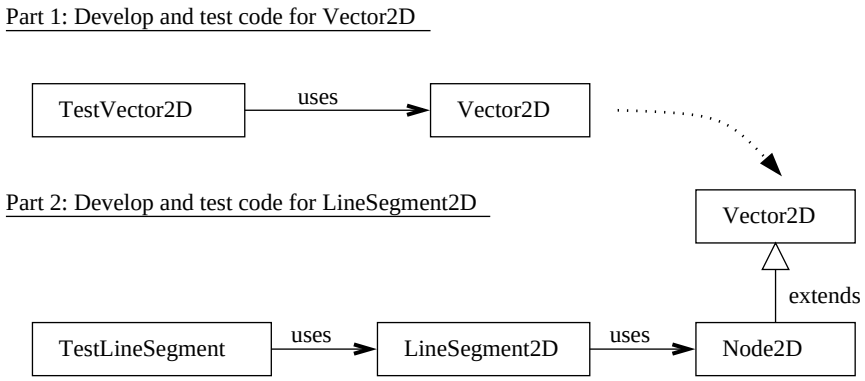

Figure 5: Schematic of test programs and arrangement of classes.

First, we develop and test Vector2D (see the upper section of Figure 5). The second test program

exercises the Node2D and LineSegment2D classes by assembling the arrangement of line segments (i.e, see $s1$ and $s2$) and data points (i.e, see $pt1$, $pt2$, $pt3$ and $pt4$) shown in Figure 6.

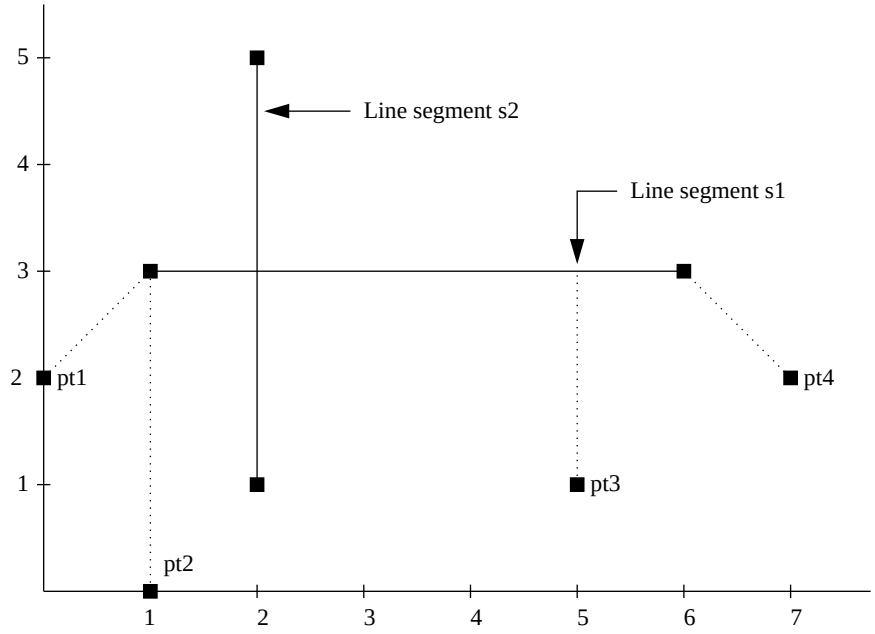

Figure 6: Two-dimensional coordinate system containing line segements and points.

Notice that points $pt1$ and $pt4$ are in the end zone; points $pt2$ and $pt3$ are in the "near line" zone.

Details of the Java source code files are as follows:

File 1: Vector2D.java

source code

```
/**
  *  ========================================================
  *  Vector2D.java: Source code for two-dimensional vectors.
  *
  *  Written by: Mark Austin                          April, 2023
  *  ========================================================
  */

package geometry;

import java.lang.Math;

public class Vector2D {
   protected double dX, dY;

   // Constructor methods ....
```

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

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

// Retrieve x and y coordinates ...

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

// Compute magnitude of vector ....

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

// Vector Arithmetic: Add vector v1 to v .....

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

// Vector Arithmetic: Subtract vector v1 from v .....

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

// Vector Arithmetic: 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 to one ...

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;
   }

   // Create string representation of vector ...

   public String toString() {
      String s = String.format("Vector2D( %7.3f, %7.3f )", dX, dY );
      return s;
   }
}
```

File 2: Node2D.java

source code

```
/**
  *  ============================================================
  *  Node2D.java: Java class for nodes in two dimensions.
  *
  *  Written by: Mark Austin                          April, 2023
  *  ============================================================
  */

package geometry;

public class Node2D extends Vector2D {
   private String name;

   // Constructor methods ....

   public Node2D() {
      super( 0.0, 0.0 );
   }

   public Node2D( double dX, double dY ) {
      super( dX, dY );
   }

   public Node2D( String name, double dX, double dY ) {
      super( dX, dY );
      this.name = name;
   }

   // Set name for the node ...

   public void setName( String name ) {
      this.name = name;
```

```
   }

   // Convert node to a string ...

   public String toString() {
      return "Node2D(\"" + name + "\") is at (" + dX + "," + dY + ")";
   }
}
```

File 3: LineSegment2D.java

source code

```
/*
 *  ========================================================================
 *  LineSegment2D.java: A line segment is defined by the (x,y) coordinates
 *                      of its two end points.
 *
 *  Written By: Mark Austin                                          April 2023
 *  ========================================================================
 */

package geometry;

import java.lang.Math;

public class LineSegment2D {
   private Node2D n1, n2; // nodal points defining the line segment
   private String name = null;

   // Constructor methods ...

   public LineSegment2D() { }

   public LineSegment2D( Node2D n1, Node2D n2 ) {
      this.n1 = n1;
      this.n2 = n2;
   }

   public LineSegment2D( double dX1, double dY1, double dX2, double dY2 ) {
      n1 = new Node2D( dX1, dY1 );
      n2 = new Node2D( dX2, dY2 );
   }

   // Get end points n1 and s2.

   public Node2D getNode1() {
      return n1;
   }

   public Node2D getNode2() {
```

```
   return n2;
}

// Set/get line segment name ...

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

public String getName() {
   return name;
}

// Compute length of line segment.

public double getLength() {
   double dLength;

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

   return Math.sqrt(dLength);
}

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

public double distanceToPoint( Node2D 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() );
}

// Create string representation of line segment ...

public String toString() {
   StringBuffer buf = new StringBuffer();

   if ( name == null )
      buf.append("Line Segment \n");
   else
```

```
         buf.append( String.format( "Line Segment: %s ...\n", name ));

      buf.append(String.format("--- node 1: (x,y) = (%.3f, %.3f) ...\n", n1.getX(), n1.getY() ));
      buf.append(String.format("--- node 2: (x,y) = (%.3f, %.3f) ...\n", n2.getX(), n2.getY() ));
      buf.append(String.format("--- length = %8.2f ...\n", getLength() ));

      return buf.toString();
   }
}
```

Test Program for Vector2D. The test program for Vector2D.java, i.e.,

source code

```
/**
  *  ============================================================
  *  TestVector2D.java: Exercise methods in Vector2D class ...
  *
  *  Written by: Mark Austin                                April, 2023
  *  ============================================================
  */

package demo;

import geometry.*;

import java.lang.Math;

public class TestVector2D {
   public static void main ( String args[] ) {

      System.out.printf("Vector2D test program ...\n");
      System.out.printf("============================================== \n");

      // Create two test vectors ...

      Vector2D vA = new Vector2D( 1.0, 2.0 );
      Vector2D vB = new Vector2D( 2.0, 2.0 );

      // Print test vectors ...

      System.out.println( "Vector2D vA = " + vA.toString() );
      System.out.println( "Vector2D vB = " + vB.toString() );

      // Compute and print sum and difference of test vectors ...

      System.out.println( "Vector2D vA + vB = " + vA.add(vB).toString() );
      System.out.println( "Vector2D vA - vB = " + vA.sub(vB).toString() );

      // Compute and print normalized test vectors ...

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

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

      // Verify that dot products vA.vB and vB.vA are equal ...

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

      System.out.printf("============================================== \n");
      System.out.printf("Done! ... \n");
   }
}
```

---

generates the output:

```
geom01:
     [java] Vector2D test program ...
     [java] ==============================================
     [java] Vector2D vA = Vector2D(   1.000,   2.000 )
     [java] Vector2D vB = Vector2D(   2.000,   2.000 )
     [java] Vector2D vA + vB = Vector2D(   3.000,   4.000 )
     [java] Vector2D vA - vB = Vector2D(  -1.000,   0.000 )
     [java] vA.normalize() = Vector2D(   0.447,   0.894 )
     [java] vB.normalize() = Vector2D(   0.707,   0.707 )
     [java] Dot product vA.vB = 6.0
     [java] Dot product vB.vA = 6.0
     [java] ==============================================
     [java] Done! ...
```

Test Program for LineSegment2D. The test program for LineSegment2D.java is as follows:

source code

```
/*
 *  ========================================================================
 *  TestLineSegment2D.java: Exercise methods in line segment class.
 *
 *  Written By: Mark Austin                                        April 2023
 *  ========================================================================
 */

package demo;

import geometry.*;

import java.lang.Math;

public class TestLineSegment2D {
   public static void main( String args[] ) {
   double dX, dY;
```

```
      System.out.printf("LineSegment2D test program ...\n");
      System.out.printf("============================================== \n");

      // Create line segment s1 ...

      LineSegment2D s1 = new LineSegment2D( 1.0, 3.0, 6.0, 3.0 );
      s1.getNode1().setName("n1");
      s1.getNode2().setName("n2");
      s1.setName("s1");

      // Create line segment s2 ...

      Node2D n1 = new Node2D( 2.0, 1.0 );
      n1.setName( "n3" );
      Node2D n2 = new Node2D( 2.0, 5.0 );
      n2.setName( "n4" );

      LineSegment2D s2 = new LineSegment2D( n1, n2 );
      s2.setName("s2");

      // Print details of line segments.

      System.out.println ( s1.toString() );
      System.out.println ( s2.toString() );

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

      System.out.printf("Compute distance of a point to line s1 ... \n");
      System.out.printf("\n");

      Node2D pt1 = new Node2D ( "pt1", 0.0,  2.0 );
      System.out.println( pt1.toString() + ": distance to s1 = " +
                          s1.distanceToPoint( pt1 ) );

      Node2D pt2 = new Node2D ( "pt2", 1.0,  0.0 );
      System.out.println( pt2.toString() + ": distance to s1 = " +
                          s1.distanceToPoint( pt2 ) );

      Node2D pt3 = new Node2D ( "pt3", 5.0,  1.0 );
      System.out.println( pt3.toString() + ": distance to s1 = " +
                          s1.distanceToPoint( pt3 ) );

      Node2D pt4 = new Node2D ( "pt4", 7.0,  2.0 );
      System.out.println( pt4.toString() + ": distance to s1 = " +
                          s1.distanceToPoint( pt4 ) );

      System.out.printf("============================================== \n");
      System.out.printf("Done! ... \n");
   }
}
```

generates the output:

```
geom02:
     [java] LineSegment2D test program ...
     [java] ================================================
     [java] Line Segment: s1 ...
     [java] --- node 1: (x,y) = (1.000, 3.000) ...
     [java] --- node 2: (x,y) = (6.000, 3.000) ...
     [java] --- length =     5.00 ...
     [java]
     [java] Line Segment: s2 ...
     [java] --- node 1: (x,y) = (2.000, 1.000) ...
     [java] --- node 2: (x,y) = (2.000, 5.000) ...
     [java] --- length =     4.00 ...
     [java]
     [java] Compute distance of a point to line s1 ...
     [java]
     [java] Node2D("pt1") is at (0.0,2.0): distance to s1 = 1.4142135623730951
     [java] Node2D("pt2") is at (1.0,0.0): distance to s1 = 3.0
     [java] Node2D("pt3") is at (5.0,1.0): distance to s1 = 2.0
     [java] Node2D("pt4") is at (7.0,2.0): distance to s1 = 1.4142135623730951
     [java] ================================================
     [java] Done! ...
```

Please look at the source code carefully, and then answer the following questions:

PART 1: Develop and test code for Vector2D.

[**2a**] (5 pts). What are the two types of comment statement used in the Vector2D source code?

[**2b**] (5 pts). What does the line:

```
import java.lang.Math;
```

do in the Vector2D source code? And why is it included in this program?

[**2c**] (5 pts). List the methods whose source code is contained within Vector2D.

[**2d**] (5 pts). The declaration for variables dX and dY in Vector2D is:

```
public class Vector2D {
   protected double dX, dY;
```

What constraints on access to dX and dY are imposed by use of the keyword `protected`?

[**2e**] (5 pts). The pair of lines:

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

generates the output:

```
Dot product vA.vB = 6.0
Dot product vB.vA = 6.0
```

Explain how this computation works.

PART 2: Develop and test code for LineSegment2D.

[**2f**] (10 pts). Draw and label a diagram that shows the organizational arrangement of Java packages and user-defined Java source code files for the classes shown in Figure 5.

[**2g**] (5 pts). The point-to-line-segment distances are computed and printed with statements of the form:

```
System.out.println( pt1.toString() + ": distance to s1 = " + s1.distanceToPoint( pt1 ) );
```

Briefly explain how the string argument to println() is systematically assembled.

[**2h**] (10 pts). Draw and label a diagram that shows the layout of memory generated by the fragment of code:

```
// Create line segment s2 ...

Node2D n1 = new Node2D( 2.0, 1.0 );
n1.setName( "n3" );                              <--- Part 1.

Node2D n2 = new Node2D( 2.0, 5.0 );
n2.setName( "n4" );                              <--- Part 2.

LineSegment2D s2 = new LineSegment2D( n1, n2 );
s2.setName("s2");                                <--- Part 3.
```

I suggest that you divide your explanation into three parts.