OOP - Spring 2015 - Review Exercises

1. Write a complete Java program that prompts the user for the name of an input file, and displays on console the number of lines in the file. Assume that you are giving the name of a text file as input.

2. Consider a checked Exception:

```
class myException extends Exception {
   public  myException(){
         super("Negative number supplied");
   }
}
```

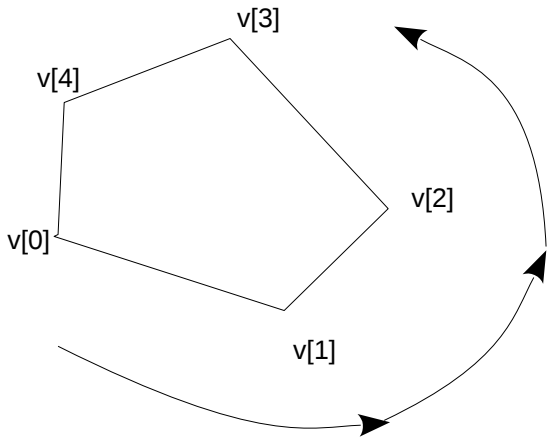
Write a program that prompts the user for a non-negative integer, and displays the square of that integer. If the input integer is negative, the program should throw the Exception listed above and quit.

3. Write a complete Java program that prompts the user for the name of an output file, and writes to that file the squares of 1, 2, 3, 4 respectively. If the file could not be written to, a single line message should be displayed saying "Exception writing to file"

4. Write a Java program using the Stack data structure that examines a given input text file and lists any un-matched '(', '{' or '[' occurring in the file.

5. In this exercise, you are constructing a class hierarchy of shapes slightly different from the ones we discussed in class. You still have the same Shape interface, and a Point class that has an extra static method to return the distance between two points, *static double dist(Point p1, Point p2).*
The expression to use is this: If (x1, y1) and (x2, y2) are two points in the plane, the distance between them is $\sqrt{(x1-x2)^2+(y1-y2)^2}$

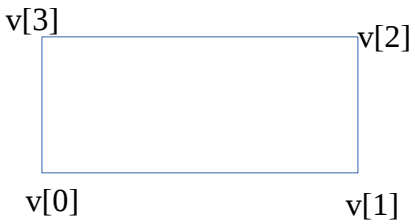
In geometry, a convex polygon of *n* sides will have *n* vertices (prove this!). We define the position of the polygon as the first vertex position, and we assume that the vertices are arranged in anti-clockwise sense, starting at the first vertex.



A 5-sided polygon
numSides=5
Vertices[]={v[0], v[1], v[2], v[3], v[4]}
position=v[0]

The abstract class Polygon implements Shape. It has an array of Points called *Vertices*, and an int, *numSides*, the number of sides of the polygon, which is set in the constructor as the length of the array of vertices. The constructor takes an array of Points as argument to construct the Polygon - it sets the *numSides* to the length of the supplied array, and also sets the *Vertices* to this array. Polygon class also implements getPosition() method, which simply returns *Vertices[0]*.

The Rectangle class extends Polygon, has 4 sides and 4 vertices (actually 3 vertices will determine the rectangle completely).



The *getArea()* in Rectangle class returns the area which is the product of the distances from v[0] to v[1] and v[1] to v[2]. Use the distance between two Points method in Point class (*dist* method) to find the area. *getName()* returns "Rectangle". The Rectangle constructor takes an array of 4 Point objects and calls the Polygon's constructor.

See figure below following the code.

a. Why is it necessary for Polygon to be abstract even though it has no abstract methods?

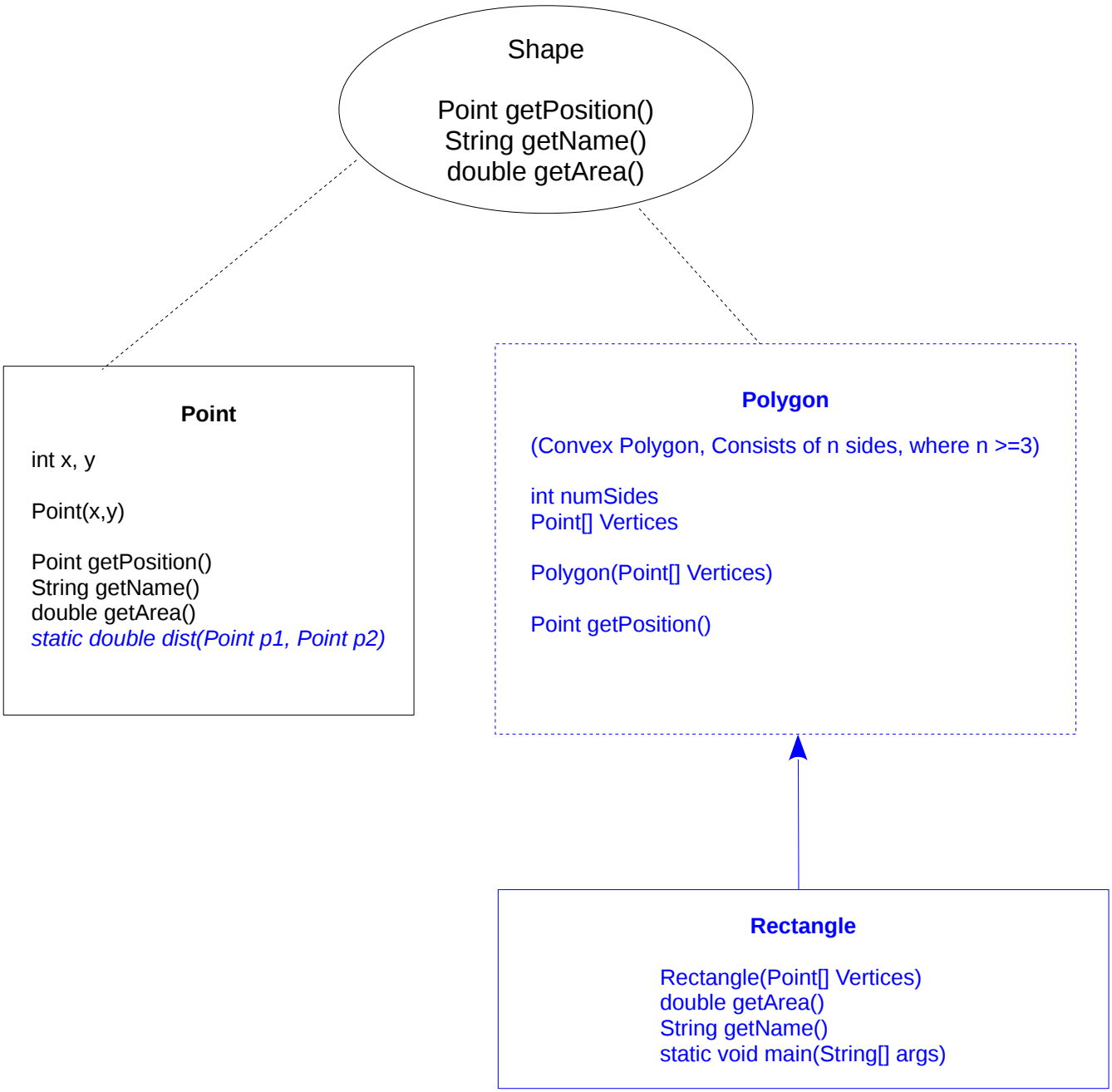
b. Fill in the code fragments to complete the program, indicated in blue. Run the program. You must leave all code unchanged, except the ones marked in blue.

```
/** Interface defining the behavior common to all shapes
  */
interface Shape {
   /** Retrieve the position of the shape
     * @return the shape's position
     */
   public Point getPosition();

   /** Retrieve the name of the shape
     * @return the name of the shape
     */
   public String getName();

   /** Calculate the area covered by the shape
     * @return the shape's area
     */
   public double getArea();
}
class Point implements Shape {
   private int x, y;
   /** Construct a point with an X and Y position
     * @param x1 X position of the point
     * @param y1 Y position of the point
     */
   public Point( int x, int y ) {
```

```
    this.x = x;
    this.y = y;
  }
  /** Retrieve the position of the point
   * @return the point's position
   */
  public Point getPosition() {
    // a point is its own position
    return this;
  }

  /** Retrieve the name of the point
   * @return the name of the point
   */
  public String getName() {
    return "point";
  }

  /** Calculate the area covered by the point
   * @return the point's area.  This will always
   * be 0.
   */
  public double getArea() {
    return 0;
  }

   /** return the distance between two Points */
    public static double dist(Point p1, Point p2){
            //your code

    }
}

abstract class Polygon implements Shape{
   /**
    * numSides is the number of sides in a polygon
   */
   int numSides;
   /*
    * Vertices[] is the array of vertices, listed in counter-clockwise sense
    */
   Point[] Vertices;
   public Polygon(Point[] Vertices){
            //your code


   }
   /**
    * We define position of a polynomial as the Point of the first vertex
    */
   public Point getPosition(){
            return this.Vertices[0];
   }
   /**
    * The following are optional
    */
   //public abstract String getName();
   //public abstract double getArea();
```

```
}

class Rectangle extends Polygon {
   public Rectangle(Point[] Vertices){
          //your code

   }
   public double getArea(){
      //your code

   }
   public String getName(){
          return "Rectangle";
   }
   public static void main(String[] args){
          Point p1 = new Point(1,4);
          Point p2 = new Point(7,4);
          Point p3 = new Point(7,10);
          Point p4 = new Point(1,10);
          Point[] pA={p1,p2,p3,p4};
          Rectangle R1=new Rectangle(pA);
          System.out.println("Name = " +R1.getName());
          System.out.println("Area = " + R1.getArea());
   }
}
```

```
/**OUTPUT **

Name = Rectangle
Area = 36.0
**********/
```


Shape
Point getPosition()
String getName()
double getArea()
Point
int x, y
Point(x,y)
Point getPosition()
String getName()
double getArea()
static double dist(Point p1, Point p2)
Polygon
(Convex Polygon, Consists of n sides, where n >=3)
int numSides
Point[] Vertices
Polygon(Point[] Vertices)
Point getPosition()
Rectangle
Rectangle(Point[] Vertices)
double getArea()
String getName()
static void main(String[] args)