# Files

This will involve the following files

* + Point.java A java class you will write that represents a Point in 2d space

Polygon.java A java class you will write that represents a Polygon in 2d space (rep- resented as a series of points)

*•*

ShapeDrawer A java class that uses your point and polygon class (as well as built-in java tools) to render images of polygons. NOTE – this file won’t compile until Polygon and Point are both complete (or nearly-so)

*•*

* + Figure.java A java program (class with main method) that generates some image
  + image.png The image your Figure program generates
  + PointTest.java A provided testing file that runs through our Point class tests.
  + PolygonTest.java A provided testing file that runs through our Polygon class tests.

# Requirements and Software Design

## Point

You must make a Point class from scratch. The point class will represent a Point in 2d space by its “x” and “y” location, and must have the following public properties:

public Point(double x, double y) A two argument constructor that allows setting initial values for its x and y location.

*•*

* + - public double getX() A method that allows getting the current x value.
    - public double getY() A method that allows getting the current y value.

public void move(double dx, double dy) A method that moves the point by a specified amount (so move the point by dx on the x axis and dy on the y axis) Note – this should be *relative* movement – based on its current location.

*•*

public void rotateAroundOrigin(double theta) A method that rotates the point about the origin of the coordinate system (0*,* 0) The update equation for this transfor- mation is as follows:

*•*

*xnew* = *x × cos*(*θ*) *− y × sin*(*θ*)

*ynew* = *y × cos*(*θ*) + *x × sin*(*θ*)

The theta value will be given in degrees. Make sure to check whether the java Math functions take degrees or radians. If you need to convert between degrees and radians, make sure your conversion is precise enough, or else you may get an incorrect answer.

public String toString() The function should return a String representation of the current object. The format for this should be based on the following example (1.2, 34.0) (so parenthesis, x value, comma space, y value, close parenthesis)

*•*

public static double distance(Point p1, Point p2) This class method should compute the distance between two points:

*•*

*distance* = ✓(*x*1 *− x*2)2 + (*y*1 *− y*2)2

## Polygon

You must make a Polygon class from scratch. The polygon class will represent a polygon (general term for any 2d shape) as a series points. The idea is that the points would be connected in a cycle (first point connects to second point, second to third and so-forth, until finally the last point connects to the first point to close the shape) So a square might look like this (0.0, 0.0) -- (2.0, 0.0) -- (2.0, 2.0) -- (0.0, 2.0) with the final edge

((0.0, 2.0) -- (0.0, 0.0) being implied)

There are other ways to represent this series of points in java, and you are free to choose private properties as you see fit, however, I recommend using an array of Point objects to store the series of points used to represent this class.

The Polygon class must have the following public properties:

Polygon(Point[] points) A constructor that provides the number, and initial loca- tion of the series of points that make the shape by providing an array of points. You can assume this array is not null, and that all positions in the array are also not null

*•*

* + - int getNumPoints() this should indicate the number of points in the polygon

Point getPoint(int index) this should return the point in the specified location in the sequence of points. If the index is invalid (greater-than or equal-to the number of points, or negative) then the special value null should be returned.

*•*

public Point getCenter() This should return a point that represents the center of the shape. We will compute this by taking the average x value of each point in the shape, and the average y value of each point in the shape.

*•*

public double getPerimeter() This should return the length of the perimeter of the shape. This can be computed as the distance between each sequential pairs of points. Don’t forget the line connecting the first and last point.

*•*

public String toString() This should return a string representation of the shape. An example of this format can be seen earlier:

*•*

(0.0, 0.0) -- (2.0, 0.0) -- (2.0, 2.0) -- (0.0, 2.0). Formally, this should

be formed as per the point toString, with each point separated by a space, two dashes, then another space.

public void move(double dx, double dy) this should move the shape. This will have the effect of moving each point in the shape by the listed amounts

*•*

public Point getTopLeft() this should return a point indicating the top-left loca- tions used in the shape. Note, this is not guaranteed to be one of the points that make up the shape. In fact, frequently this will not be a point in the shape. Instead, it should return a point whose x value is the left-most (minimum) x value of any point in the shape and whose y value is the top-most (minimum) y value. (Yes, I’m aware that’s a little counter-intuitive, computer graphics often makes ”down” mean ”larger y values” despite normal practice in the field of mathematics)

*•*

public Point getBottomRight() this should return a point indicating the bottom- right location used in the shape. Again – this likely won’t be in a point in the shape itself. Instead it should have the maximum x and y values used in the shape.

*•*

public void rotateAroundCenter(double theta) This should rotate the shape around it’s center. This will have a side-effect of moving each point in the shape to a new lo- cation, but should not have a meaningful effect on the center of the shape. **HINT** You may find it easiest to implement this by using several other functions from the two classes you’ve built.

*•*

## ShapeDrawer

You do not need to program this class, but you are expected to download the class from canvas and read the javadocs to figure out how to use it. This class will not compile until both Point and Polygon are nearly complete – so you may wish to avoid downloading this file until you’re down with those classes.

When used properly, this class will allow you to render images based on polygons.

## Figure

The Figure class should have one method: public static void main(String[] args). This program should use the ShapeDrawer, as well as your Point and Polygon classes and create an image “image.png”. Use the following line of code (assuming drawer is your Shape- Drawer) to render the image. This should ensure the file is created where the autograder will look for it.

drawer. writeTo File(" image. png");

Your figure must:

Show at least one polygon that has at least 5 points (more polygons are fine, as are more points. So long as at least one polygon has at least 5 points you’re good)

*•*

* + - Your code should use one-of the move, or rotate functions on the polygon.
    - The image should be at least 100 pixels wide and 100 pixels tall.

Beyond these requirements you are free to do what you wish. Make sure this class runs correctly and submit your image.png to demonstrate that you’ve run the code correctly. As a few ideas:

Try using a for loop, and some equation to create a shape. With some trigonometry This can be great for “simple” shapes like squares or regular polygons, but you can also get weird results like stars with the right equations

*•*

* + - Try repeatedly moving and drawing one shape
    - See if you can figure out how to choose custom colors
    - See if you can include random behavior a bit

**Submission**

For this we expect 4 files to be submitted:

* + Figure.java
  + Point.java
  + Polygon.java
  + image.png (the image you generated from Figure.java on your computer)