2.1 Written Report

1. For each of the programs keep track of how much time you spend designing, coding and correcting errors, and how many errors you need to correct. Sum these numbers once your implementation is complete.

The solution was designed, implemented, tested and refined over a period of 34 days (02/04/2021 - 06/05/2021). Code was written on 10 of those days.

Designing/Research:

Date	**What**	**Time**
02/04/2021	Program structure	0.5 hours
04/04/2021	Shape inheritance hierarchy	0.5 hours
08/04/2021	File reading with regex	1 hours
08/04/2021	Shape factory method	0.5 hours
10/04/2021	CompareTo() and sorting	0.5 hours
11/04/2021	LL Iterator	1 hours
	Total:	4 hours

Coding:

Date	**What**	**Time**
02/04/2021	- Create Git repo - Copy relevant code form A1 to A2	0.5 hours
04/04/2021	- Write initial classes for shapes	1 hour
08/04/2021	- File reading	2 hours
08/04/2021	- Make LL generic and test	2 hours
08/04/2021	- Shape factory	0.5 hours
08/04/2021	- Area methods and test	2 hours
09/04/2021	- originDistance() methods	2 hours

10/04/2021	- CompareTo() and sorting	2.5 hours
11/04/2021	- LL Iterator	0.5 hours
12/04/2021	- File level comments	0.5 hours
13/04/2021	- Method level comments	1 hours
16/04/2021	- Line level comments	1 hours
	Total:	15.5 hours

Error Correction:

Date	**Error**	**Solution**	**Source**	**Time**
02/04/2021	- New node is implicitly a sentinel	- Explicitly make sentinel	Design	0.5 hours
08/04/2021	- LL could store everything	- Make it only store objects that extend PlanarShape	Design	0.5 hours
08/04/2021	- LL take() method broken	- Rewrite the method	Coding	0.5 hours
08/04/2021	- File reading doesn't work for 'harder' file	- Use regex as delimiter and just read till next delimiter	Design	1 hours
09/04/2021	- SemiCircles don't sort correctly	- Implement originDistance() method in SemiCircle class (forgot too)	Coding	1.5 hours
11/04/2021	- LL Iterator misses item at head	- Make iterator start in correct position	Design	0.5 hours
			Total:	4.5 hours

Total time spent: 24 hours

2. Keep a log of what proportion of your errors come from design errors and what proportion from coding/implementation errors. Address any trends you noticed from your logs and results.

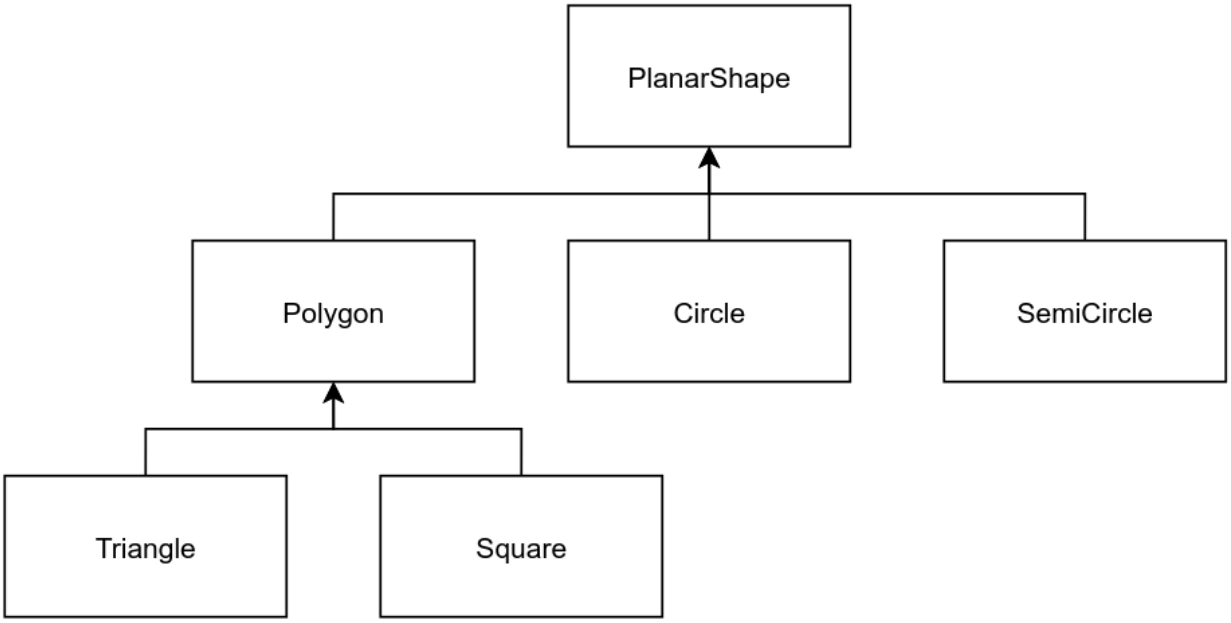
In terms of the raw amount of errors, 50% came from design errors, and 50% from code implementation errors.

But in terms of time spent, 44.4% of time spent fixing errors was fixing design errors, and 55.6% fixing coding errors.

Overall, 19% of my time on this assignment was spent fixing errors.

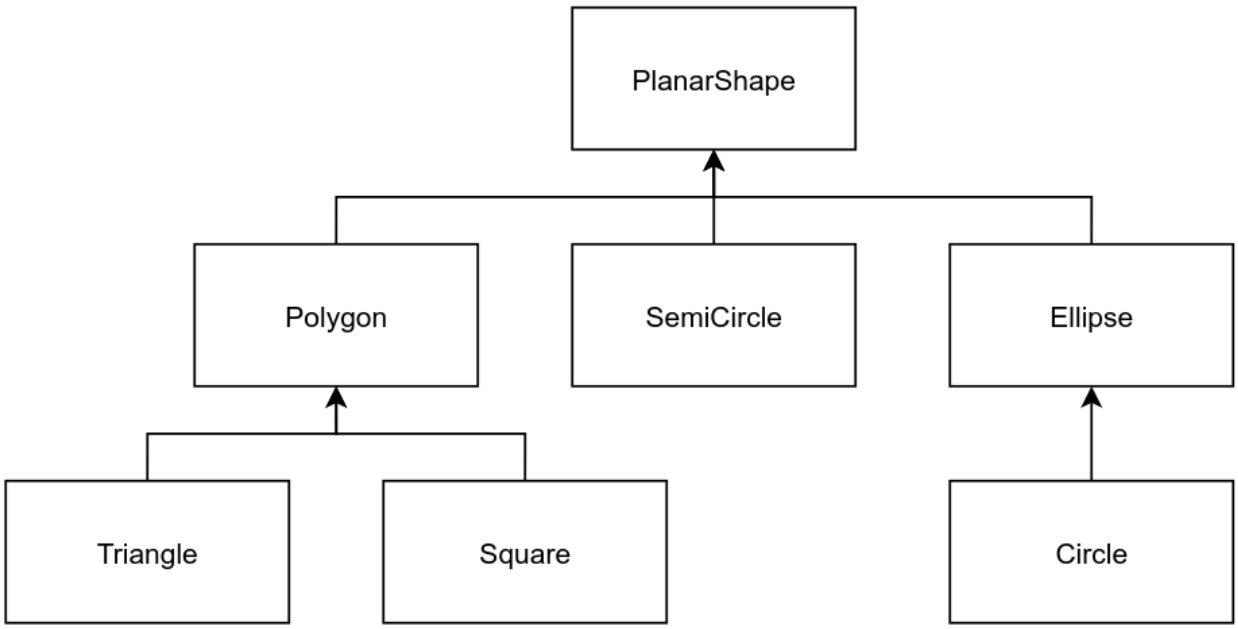
It appears to be a common trend in my assignments where the total time spent could be greatly reduced by spending more time making better design choices.

3. Provide a (brief) design of how you would further extend your PA2 so that it specifically included Triangle and Square figures. Draw the UML class diagram for this new program (intricate detail not required). What attribute data do you need in each case?



To extend PA2 by providing the functionality to store and sort triangles and squares, a Triangle and Square class will need to be created that extend the Polygon class. The Triangle and Square class are able to use the Polygon area() and originDistance() methods so they will not need to be overwritten, only toString() will need to be overwritten to provide a string representation specific to each shape (however the polygon toString() would still be valid). Triangle and Square classes would not need any additional attributes, only a limit to the number of points it can store, and functionality to check the validity of the shape (eg a square is a valid square if 4 equal length sides and 4 90 degree angles). Ideally, Square would extend a Rectangle class that extends the Polygon class instead of the Polygon class directly.

4. Investigate the mathematical structure of an Ellipse on the Cartesian plane. How would you model the Ellipse? How would you then calculate its area and originDistance()? How would this be incorporated into your program? Draw another UML class diagram to show this.



A circle is just an ellipse with ellipticity equal to 1, therefore, to incorporate the functionality to store and sort ellipses, an Ellipse class is to be created that extends PlanarShape, Circle class then extends Ellipse as seen in class diagram above.

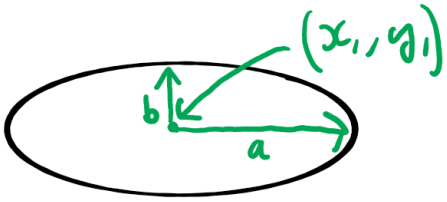
The Ellipse class should store a single cartesian point, and its semimajor and semiminor axis length. The area() method can then easily calculate the area of the ellipse by

Area = semimajor axis x semiminor axis x pi

To calculate the originDistance(), first the angle between the origin and the centre of the ellipse is to be calculated, this angle can then be used with the parametric equation of an ellipse to find the closest point on the ellipse to the origin.

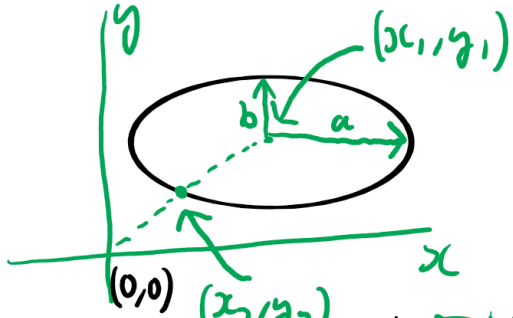
Cartesian eq^{n} of ellipse:

$$x = x_1 + a\cos(t) \sim (1)$$
$$y = y_1 + b\sin(t) \sim (2)$$



Angle between origin and center of ellipse:

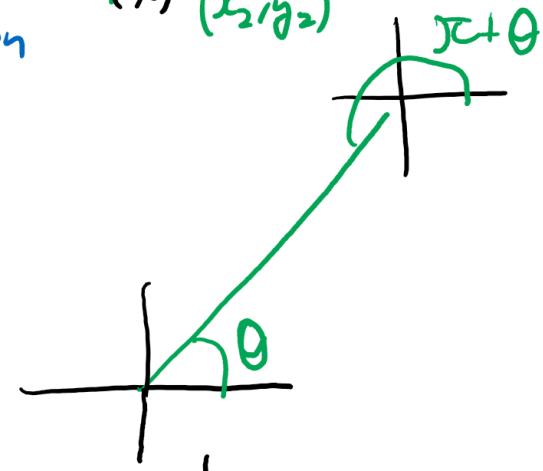
$$\theta = \text{java.lang.Math.atan2}(x_1, y_1)$$



Let (x_2, y_2) be closest point on ellipse to origin. find (x_2, y_2)

sub θ in (1) and (2):

$$x_2 = x_1 + a\cos(\pi + \theta)$$
$$y_2 = y_1 + b\sin(\pi + \theta)$$



Distance to origin:

$$d_{origin} = \left((x_2 - 0)^2 + (y_2 - 0)^2\right)^{1/2}$$