Seng 2200 Programming Language and Paradigms

SEMESTER 1, 2021 ASSIGNMENT 2:

REPORT

Stephen Watson C3339952

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.

Designing:

Reading the programs documentation took me forty minutes to understand what classes were required and to read through and highlight what classes related or connected to what classes. two hours was taken to research, understand what methods each class would require including factory method design. This was not coding content just method names and expected outcomes from the method. For example, the circle class calculateArea(), deciding if a double, float or int is required and then noting return area as the desired outcome for that method. This seems simple but helps put all pieces slowly together. Within this time, it also included putting aside what content I could keep from Assignment one. A further two hours was spent checking and researching the mathematic side of the program. Even though the documentation gave all the formulas double checking and understand of how all areas, radii and distances were going to work was paramount to have the program run successfully. A total of two hours and forty five minutes was required for the design aspect of the program.

Coding:

Initial file setup naming and saving all the files took forty minutes, this included taking everything required form assignment one and placing it into the desire class file. This gave the program a bit of structure to start building around. Below is a breakdown of each classes time to code.

Point class:

The point class took two minutes to complete as it was a direct copy from assignment one. This class took 2 minutes to save into the assignment two folder

Polygon class:

The polygon class required little coding and more shifting around of methods from assignment one. The major changes were the removal off the comparesTo() function and adding in of the originDistance() function. The following parts remand the same

- The private variables
- constructor
- toString()
- addPoint()
- CalculateArea()

This class was completed within 20 minutes

Semi-circle class:

The semi-circle class was based on the polygon class as it belongs on the same tier being a shape object. This meant all the same methods were implemented but with different mathematics required. As there was already the polygon class to base this off and previous time spent looking at the maths and methods required this class took eighty minutes to complete.

Circle class:

The circle class was based on the polygon class as it belongs on the same tier being a shape object. Once again This meant all the same methods were implemented but with different mathematics required. As there was already the polygon class to base this off and previous time spent looking at the maths and methods required this class took 60 minutes to complete.

Planar shape class:

The planer shape class is the abstract class which houses all the common functions that are shared in each shape. This class is the object a shape becomes before being stored into a node. And all the methods but one in this class are method declarations. The compareTo() from the original assignment one is the only function in this class that has implementation as the documentation says this class requires a compareTo(). To get this class correct I had a peer in the industry explain how this would work as understanding how each method for example calculateArea() would perform the correct task by calling the correct function for the correct shape. This is still a foreign concept as abstract classes have not been needed before but fairly simple to implement. This class was completed in 20 minutes.

Node class:

This was a complete copy and paste from assignment one and only required the nodes and data variables to be templated with <T>. Total time taken was fifteen minutes to complete node.

LinkedList class:

Many of the methods in the linked list class were taken from assignment one and needed to be altered for templating with <T>. The other functions were implementing the iterator class, adjusting the toString() and declaring nodes, current, length and sentinel in the correct methods.

References for iterator implementation inspired by the following sources.

- Lecture slides,
- https://www.geeksforgeeks.org/iterators-in-java/
- https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

None of these methods were not copied from the sources but gave me an understanding of what was required.

The linked list class took me one hour twenty minutes to successfully build.

Factory Method class:

The factory method class was fairly straight forward as the research into this class had a lot of information about how to set it up, what parameters to give the method and how to go about a general structure. All that was really required was matching the input stream, reading in the data, instantiating the shape objects. The main logical error here was you return the shape object at the switch but return the planar shape and the end of the method. This class took one hour to complete. Please find below sources which are also commented in my file that gave me structure for my class.

Source 1: Website

https://www.tutorialspoint.com/design_pattern/factory_pattern.htm#:~:text=Factory%20pattern%20is%20one%20of%20the%20most%20used%20design%20patterns%20in%20Java.&text=In%20Factory%20pattern%2C%20we%20create,object%20using%20a%20common%20interface.

Source 2: Website

https://www.tutorialspoint.com/design_pattern/abstract_factory_pattern.htm

Source 3: website

https://codereview.stackexchange.com/questions/122032/instantiating-shapes-using-the-factory-design-pattern-in-java

PA2 class:

The difficult part of the project was understanding what functionalities the PA2 class had and what it needed to do. After going through the design process and look at was left it gradually became clear the following was required:

- Instantiate the linked lists.
- Instantiate the factory method object.
- Read the file input and send it to the factory method class.
- Append the unordered list.
- Insert the planar shape object into the list.
- Print the lists out.

This class took 2 hours to have correct and finished.

All up this program took twelve hours of design, research, coding, organising, commenting to have it working. The report was done on top of these hours taking 3 hours. Therefore, entire project was completed in 15 hours.

Correcting errors:

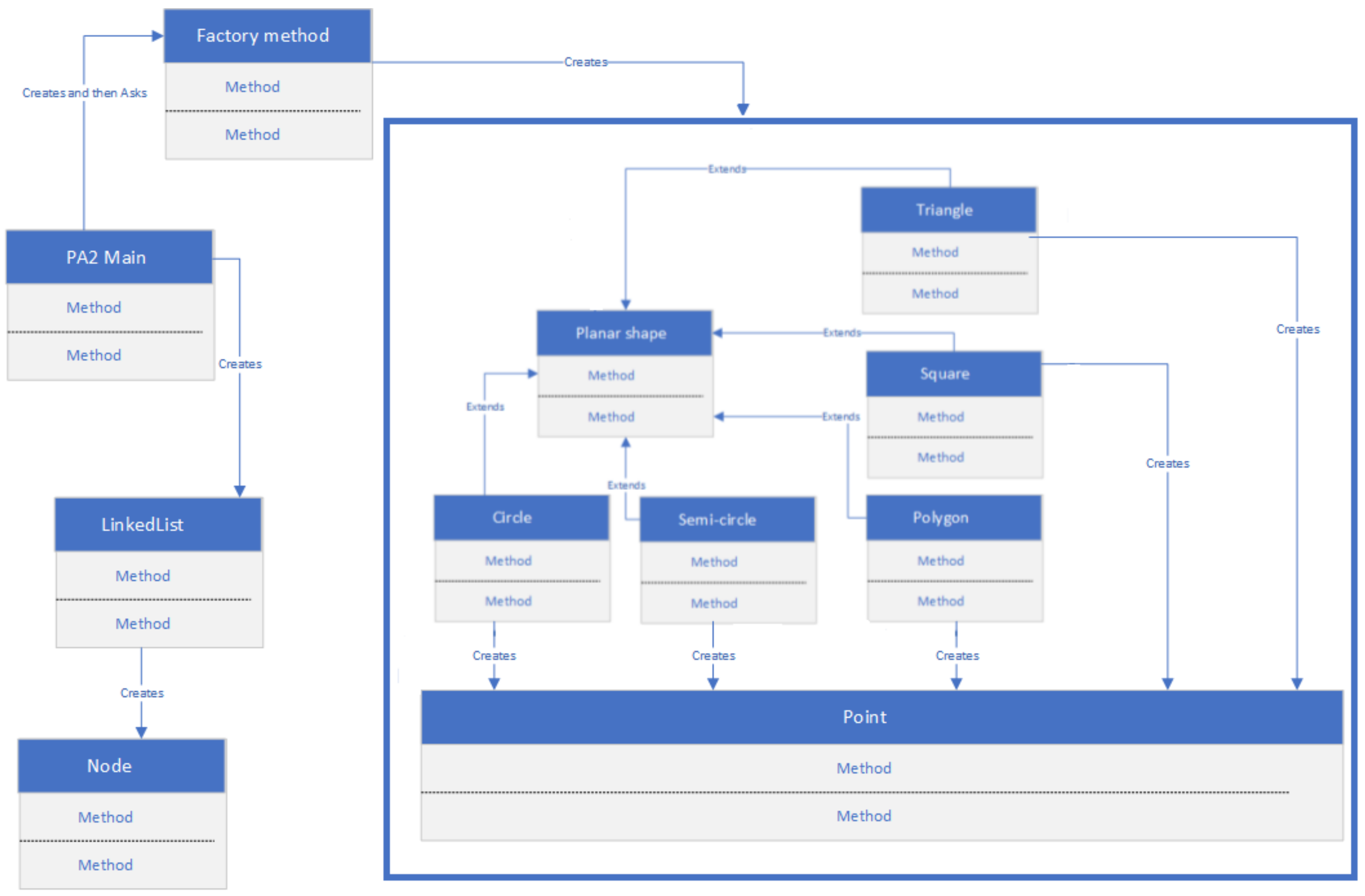
After logging errors of design and coding, fifteen logical errors were made and seventy nine syntax coding errors. Making it sixteen percent of errors were logical and eighty four percent of errors were coding, syntax errors. Based off my logs one hour and thirty minutes was used when correcting errors across all program classes.

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.

As stated above After logging errors of design and coding, fifteen logical errors were made and seventy nine coding errors. Making it sixteen percent of errors were logical and eighty four percent of errors were coding, syntax errors. Based off my logs one hour and thirty minutes was used when correcting errors across all program classes. The main trends that are showing from the logs are assignments that have strict and or very structed documentation for the required classes and methods have way more syntax errors that logical errors as a lot of the design and thinking is done or laid out for you. For example, with this program we know how the majority of classes are linked or interact. As time goes on and I gain more experience as a programmer and given less restrictions or border scope and more tools to work with, syntax errors should go down while design errors will go up as I will need to concentrate a lot more on design because the way to structure the future projects will not be laid out and will not be so clear when addressing the challenges or tasks.

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 extent the PA2 class you would need to edit the main file to read the “T” for triangle and either the “S” is only defined for semi-cirle or square, or a check is put into place to differentiate between square and semi-circle. Also have the same code included for square and triangle to process them and add into lists and append the lists. The factory method switch statement would also need to be updated to handle these values including coded to add points. Once that is done you would just need to create the shapes classes that have all the same shape methods but specifical rewritten to apply the correct mathematics for all the required formulas.



Please not this is just a simple UML diagram and classes do not connect to other classes methods. This is just to represent how the program could be expanded to add other shapes into it.

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.

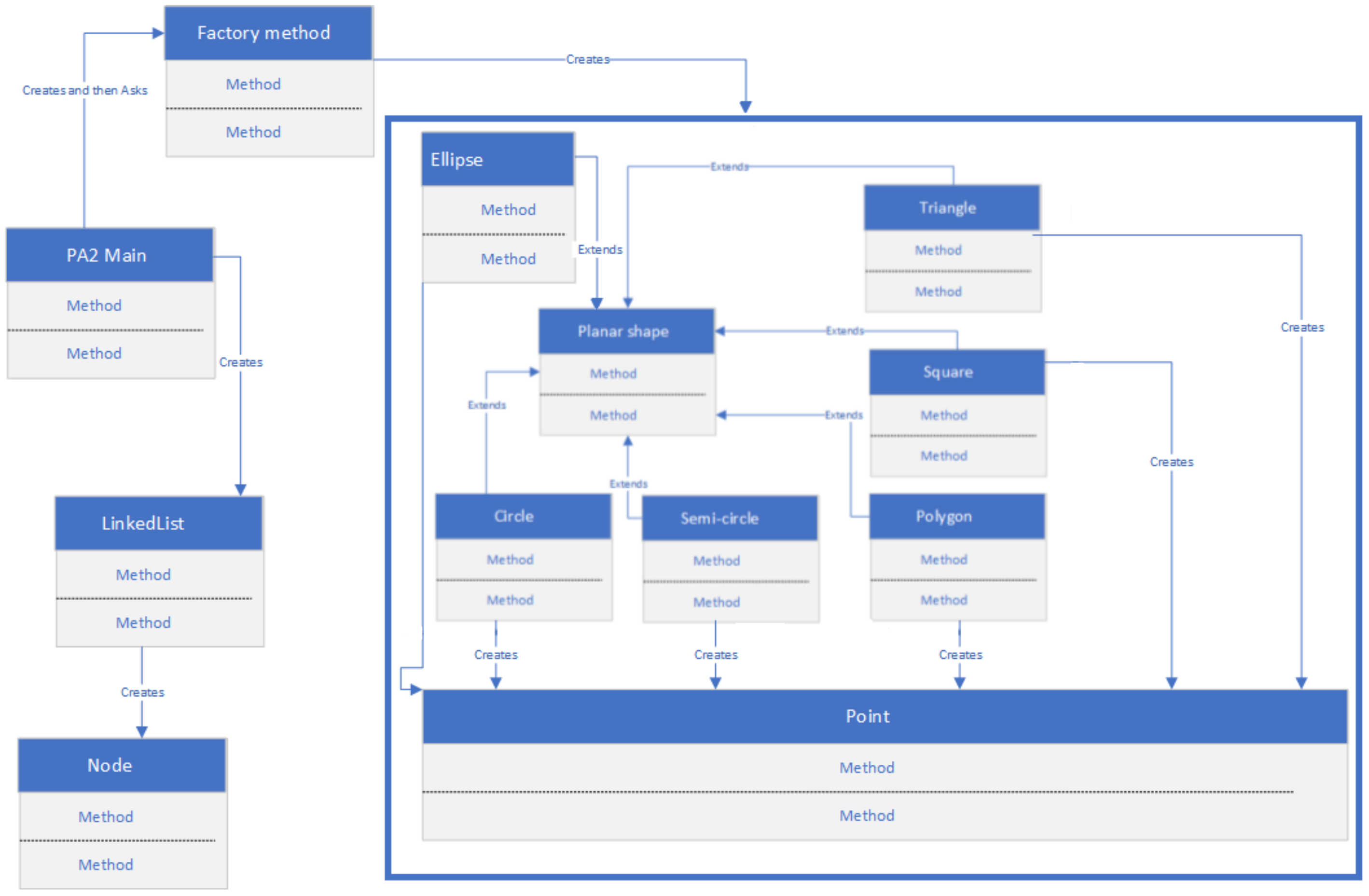
The main would need to be edited to handle a "E" value and the factory method would need to be built on to handle the "E" and process the shapes. I would use all of the same methods the other shapes do. These methods being constructor(), CalculaeArea(), originDistance(), addPoint(), toString(). Below are draft example methods used to implement an ellipse class.

Methods:

//Area of an ellipse is area = pi *a * b

a = semi major axis b = semi minor axis.

```
Calculate area()
{

   double area = (Math.PI*(a*b));

}

originDistance()

{

     //Create two new points

     double d1 = new Point(x, y);
     double d2= new Point(x, y);
     double distance =0;
     //compare distances  x1 and x2 and y1 and y2
     if(d1.x  > d2.x) return true else return false
     if( d1.y > d2.y) return true else return false
     double distance += Math.abs(d1.x – d2.x);
     distance += Math.abs(d1.y - d2.y));
     return distance;
}
```



Please not this is just a simple UML diagram and classes do not connect to other classes methods. This is just to represent how the program could be expanded to add other shapes into it. This is the UML design used to extend the PA2 class further to incorporate an ellipse. As the ellipse is just another shape The first design thought is it would just extend onto the shape tier level and just need an ellipse class file added. On further inspection the point class may need some updating to accommodate for the ellipse class. These are the current results of a simple and low level investigation and would not be able to be proven correct or incorrect until this project was taken on from design to completion.

Conclusion:

Overall, this was a great project to undertake as it gives an insight to abstract classes, the use of inheritance, polymorphism as well as a deeper understanding of linked lists which will all be daily aspects of any software engineers job.