CMSC 330 Project 1

Burkhardt, Ryan

UMGC CMSC 330

Author Note

This program reads in a file and creates a UI of hollow and solid polygons after parsing words from the .txt file. This uses many different methods of Object-Oriented Programming such as super, override and constructors as well as Lexer errors and Syntax errors.

UML Class Diagram

A diagram of a program

Description automatically generated

**Project 1 Approach**

* To start the project, I went through required documentation and extracted all necessary requirements to either add, modify, or create new classes. I then started by creating empty class files to make sure all required classes were in the project.
* With all classes required I then went into reading all source documentation that was provided such as hollow polygon, rectangle, and right triangle. After reading and analyzing what these were using as variables, I went into draw.io and created a uml with the variables I would need to create for the four classes.
* I started with Solid Polygon and copied and pasted hollow polygon into it. After reading the constructor methods I saw what the instructions were mentioning and added fill method. This was all I did in this class.
* The Isosceles Triangle I copied and pasted the right triangle code into the class. I then changed the calculations and the name of the point because we would be starting from the top center point here. After this I just changed the calculations for the points to form the triangle. The calculations were from the top center point I need to go down the height and over both left and right by half of the width.
* Parallelogram class I used the same method as rectangle but added a point and an offset while removing the height and width. I then used the offset to make the upper left and lower right to go to their proper positions.
* Regular polygon I started with the necessary objects for the constructor color, center point, radius, and number of sides. I then created the necessary private variables for each that were necessary.
* Continuing I found it hard to find a calculation that worked so I tailored the calculation from Deinst and used it as a foundation. I then had to add another calculation to correct the rotation by removing the first triangle in odd calculations.
* After all calculations were made, I then moved to the token class to make the appropriate changes. This included adding the tokens text, quotation, parallelogram, offset, space, isosceles triangle, regular polygon, radius, sides. These were necessary for the parser to accurately find each variable in the class and send it to the appropriate builder.
* The Parser class’s approach was to essentially copy what was already done for the other two classes and add the appropriate parsing tokens. With the new tokens added it allowed the parser to have a starting point and extract the appropriate data after the token.
* Also, on parser I found that I could input a number past 255 and below 0 when attempting to make a neon color. For this I added a for loop to catch the error and display the syntax error to the user on the terminal.

**What I learned**

* Scrubbing code is essential when working with anyone else’s code. This is essential to find the understanding of the code, so you know how to implement a new class using their code.
* Using the graphics class to display the appropriate image outline and fills. This was my first time using any source of graphics and taught me essentials for drawing and filling shapes.
* I also learned many things about the color class. While I simply thought that you supplied color classes with a color as for instance .black I found that providing a 3-digit color code was much harder. This led me to finding different color codes and eventually to find an error in my code when setting red to high.
* Inheritance was another function that I learned more about. This was in the case that it was a class building on a class building on a class. Such as Solid Polygon building from polygon and the other 4 objects extending Solid Polygon.
* In the token class I learned a lot about what the enumerate function is doing. This can be shown when the function is defining a set of named constants representing different types of tokens that I would use to provide variables for.
* In the text class I learned again about the different types of draw items in the Graphics class. Drawstring rendering is a complex method to start at a point and render the string and give it criteria such as font, color, visibility, bold, italic, and so much more.
* Using Overloading for the 4 classes. While sold polygon did have the color and vertex with regular polygon and the rest, we overloaded it and added the appropriate criteria such as center point and radius.
* I also learned a good bit about geometry and using calculations to create objects. Using some from geeks for geeks to understand a parallelogram and polygon incorporating them in code took a deeper knowledge to implement them correctly.
* Trigonometry was also another functional math used in the calculations especially for finding the sides of the triangle using sin and cosign. This was the most complex part of this project and again I had to use outside resources to find a calculation for this.
* Incorporation of someone else’s code. This was huge and it made me feel like this was the most practical assignment in real life. Using someone else’s code was a challenge because I not only had to make these classes I had to tailor them to this code.

**References**

**Deinst. (2010, August 8). Calculate coordinates of a regular polygon’s vertices. Stack Overflow. https://stackoverflow.com/questions/3436453/calculate-coordinates-of-a-regular-polygons-vertices**

**CMSC 330 Project 1**

**Test Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input** | **What to Test** | **Actual Output** | **Pass/Fail** |
| Text | Run – TextTest.txt | 1. Color changes on text. 2. Location changes on text 3. Actual string changes on text | All expected output worked and displayed proper output. See screenshots Test Case 1 for actual output. | Pass |
| Isosceles Triangle | Run – TestIsoTri.txt | 1. Color changes on triangle. 2. Location changes 3. Spread of triangle using width 4. Spread of triangle using height | All expected output worked and displayed proper output. See screenshots Test Case 2 for actual output. | Pass |
| Parallelogram | Run – TestPar.txt | 1. Color changes. 2. Location changes 3. Spread of offset works | All expected output worked and displayed proper output. See screenshots Test Case 3 for actual output. | Pass |
| Regular Polygon | Run – TestRegPol.txt | 1. Color changes. 2. Location changes 3. Radius Changes 4. Number of sides Changes 5. Calculation to keep odd, sided number polygon shifts to keep it straight | All expected output worked and displayed proper output. See screenshots Test Case 4 for actual output. | Pass |
| Error Test Case Isosceles Triangle | Run – ErrorTestIso.txt | 1. Setting the color to (null, null, null) to make sure the program does not take in a parameter other than 0-255. Testing that the output should produce a syntax error. | Produced syntax error as expected as well as where the error was located. | Pass |
| Error Test Case Parallelogram | Run – ErrorTestPar.txt | 1. Testing that the parallelogram cannot have negative number input because it would be off the gui. | Produced half of the parallelogram instead of producing a value error. It does allow for negative input. | Fail |
| Error Test Case Regular Polygon | Run – ErrorTestPoly.txt | 1. Converting the radius to a double to throw a number format exception and catch a syntax error because it is an int array, but I will be passing it a double. | Did not throw number format exception instead it converted the double to a number more importantly an int and allowed for creation. | Fail |

**Screenshots**

**Test Case 1**

**A screenshot of a computer

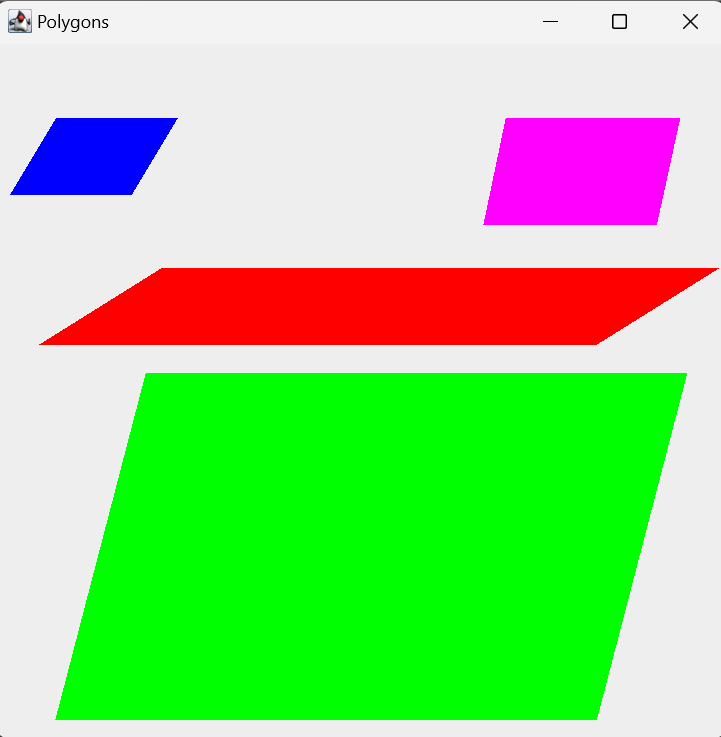
Description automatically generated**

**Test Case 2**

**A screenshot of a computer

Description automatically generated**

**Test Case 3**

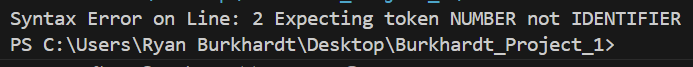
****

**Test Case 4**

**A screenshot of a computer

Description automatically generated**

**Test Case 5**

****

**Test Case 6**

**A screenshot of a computer

Description automatically generated**

**Test Case 7**

**A screenshot of a computer

Description automatically generated**