

Matthew Gregorek

CMSC330 Project 1

Spring 2023

Due: 4/11/2023

Introduction

Project 1 provides a skeleton Java program with a Word Document to help the student understand better how the program works. The objectives of this project, in my opinion, are to help students understand Context-Free Grammar (BNF) notation, provide a good example of how the Java Programming Language can be used to represent Compilation at lower-level languages, and lastly this program helps the student understand how Hierarchy in a Program works, and how using Classes, Objects and Inheritance helps the programmer create a clear and concise program.

Approach

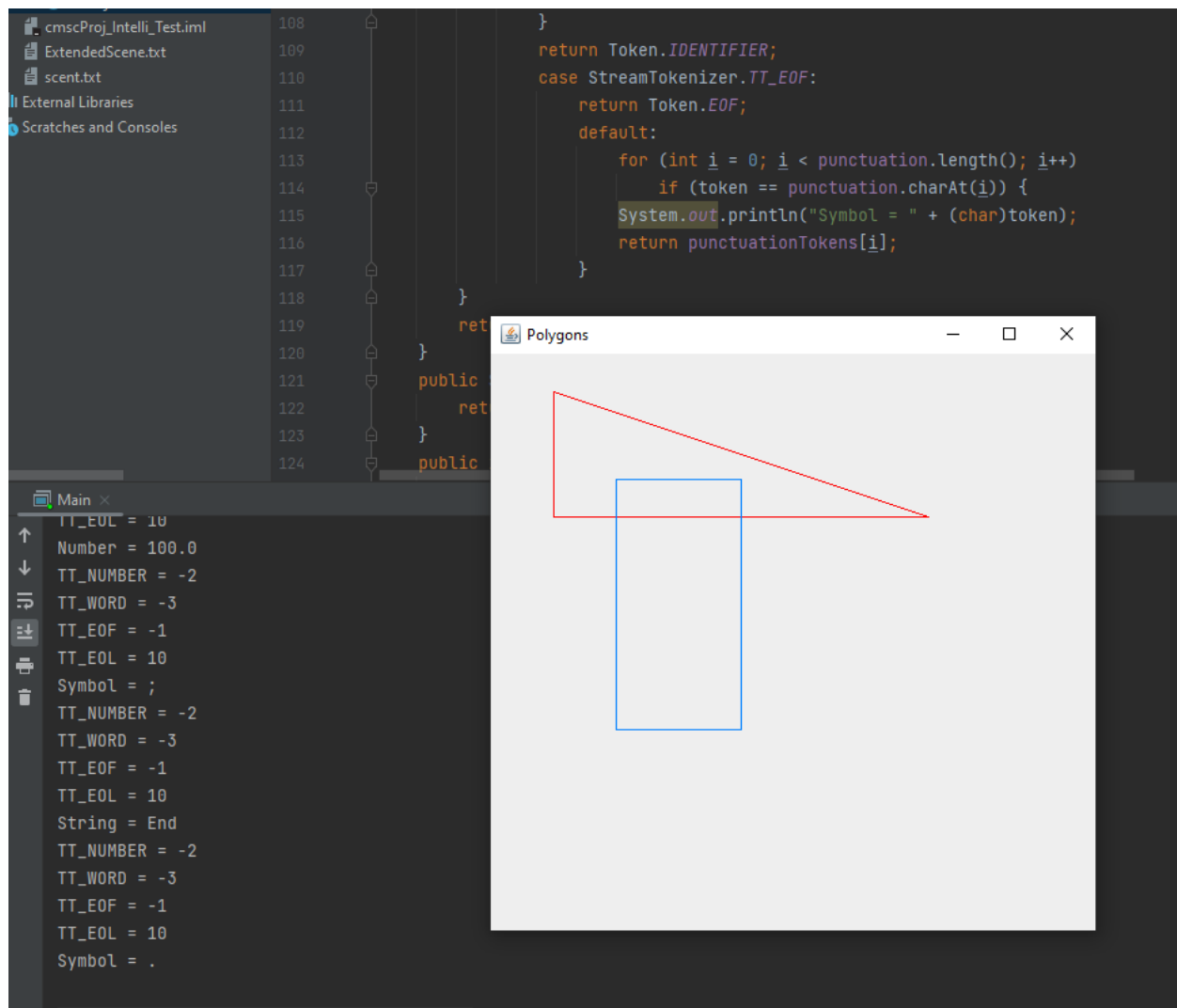
I approached this Project by downloading the skeleton zip file, and the word document file first. I unzipped the skeleton code into a project file, then imported that file into Eclipse. I was surprised to see just how much code was already provided for the project, as I have taken this class 2 times before and don't remember this project providing any code at all. So, my next approach was to read the Document File, which I guess could be considered a "readme" type of file. Realizing that this project was not necessarily going to be about my Coding skill, and more about my Programming analysis, I went over and read each Class file in depth, comparing the class file to the Hierarchy Table and the Grammar. From there, it was just a matter of tweaking some of the classes, and double checking to make sure the 3 new shapes would correspond to the SolidPolygon Class, which would then extend to the Polygon_ class, then to Image and Scene. Making sure the Enums had the Keywords and that the Parser Class would use the Lexer correctly and produce the correct type of errors also.

Learnings

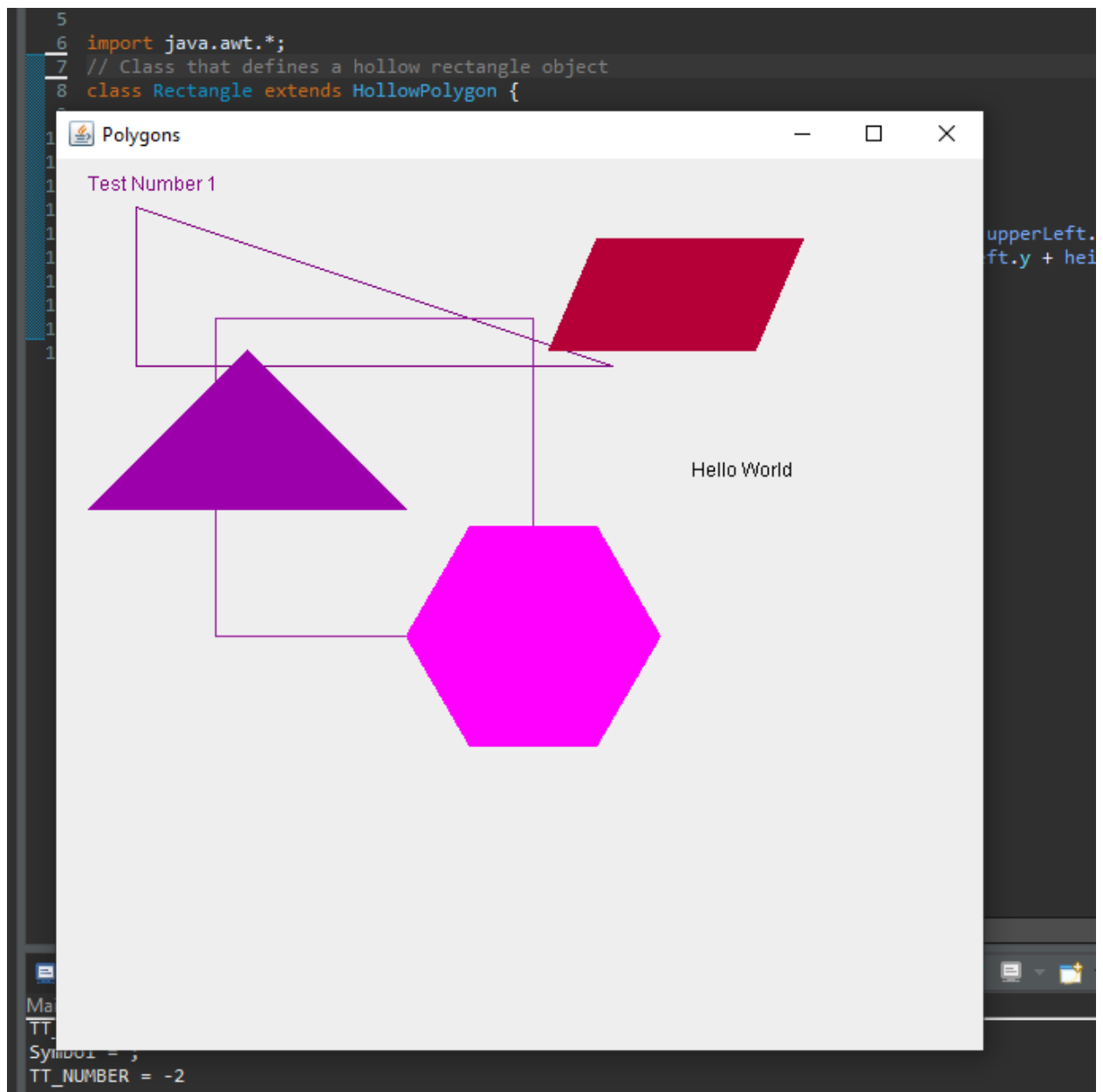
I learned that The Lexer Class uses Java's StreamTokenizer class... TT_NUMBER, TT_WORD, TT_EOF, and reads the File the user chooses, and turns each number, string, and punctuation characters into 'tokens' that will can be returned by tokenizer. I also learned that is very important to make sure that Strings, Numbers, and specific characters are declared in this class. Some of these are given negative integers, TTNUM = -2, TTWORD= -3, and TTEOF = -1, TTEOL is 10 and represent end-of-line character in ASCII, will always be shown when the tokenizer reaches end of line. This class reminded me of the GNU Lexical Analyzer I had to use for another class.

The "front-end" of this task takes input files, like scene.txt or ExtendedScene.txt, and reads the input of the text files. The getNextToken method turns the inputs into Token Enums, if the token is a number or string, it will also print out the value of it to Terminal.

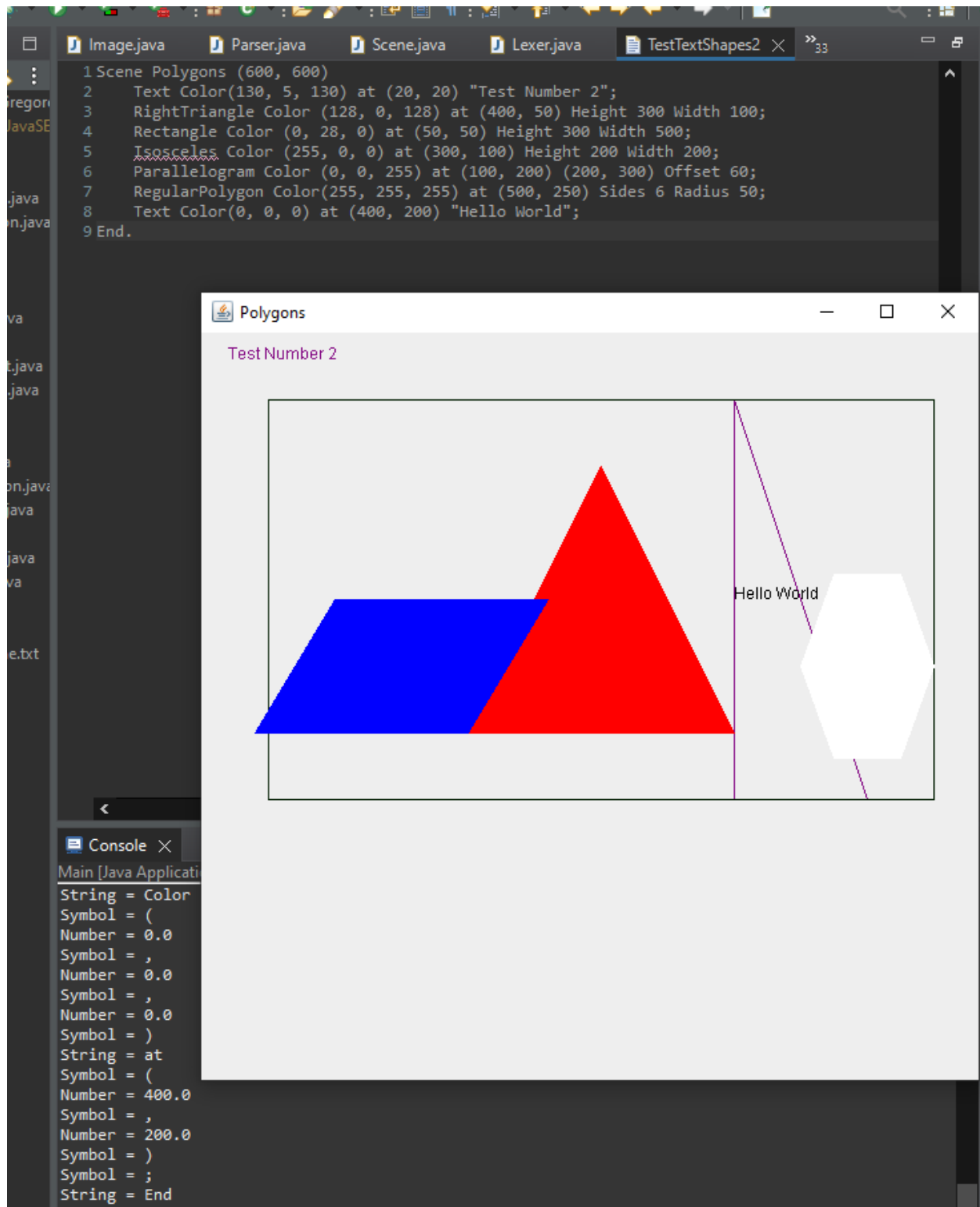
The Word Document also helped me understand that the Parser class relies on having a properly written text file to read through. If I have a missing semicolon, or a missing number, or a syntactical error, the Parser will throw the exception, whether it's Lexical error or spelling mistake. After making sure all my skeleton files were imported and no errors were showing, I ran the program, and inserted the text file 'Scene.txt' from the Word Document into my program, clicked on it, and got the correct result.



Next, I went through the rest of the document and cut out the code that it suggested I didn't need, and double checked to make sure the code for `Isosceles`, `RegularPolygon`, and `Parallelogram` Classes was correct also. Once I did that, I ran the program again using the text file `ExtendedScene` and got the correct result.

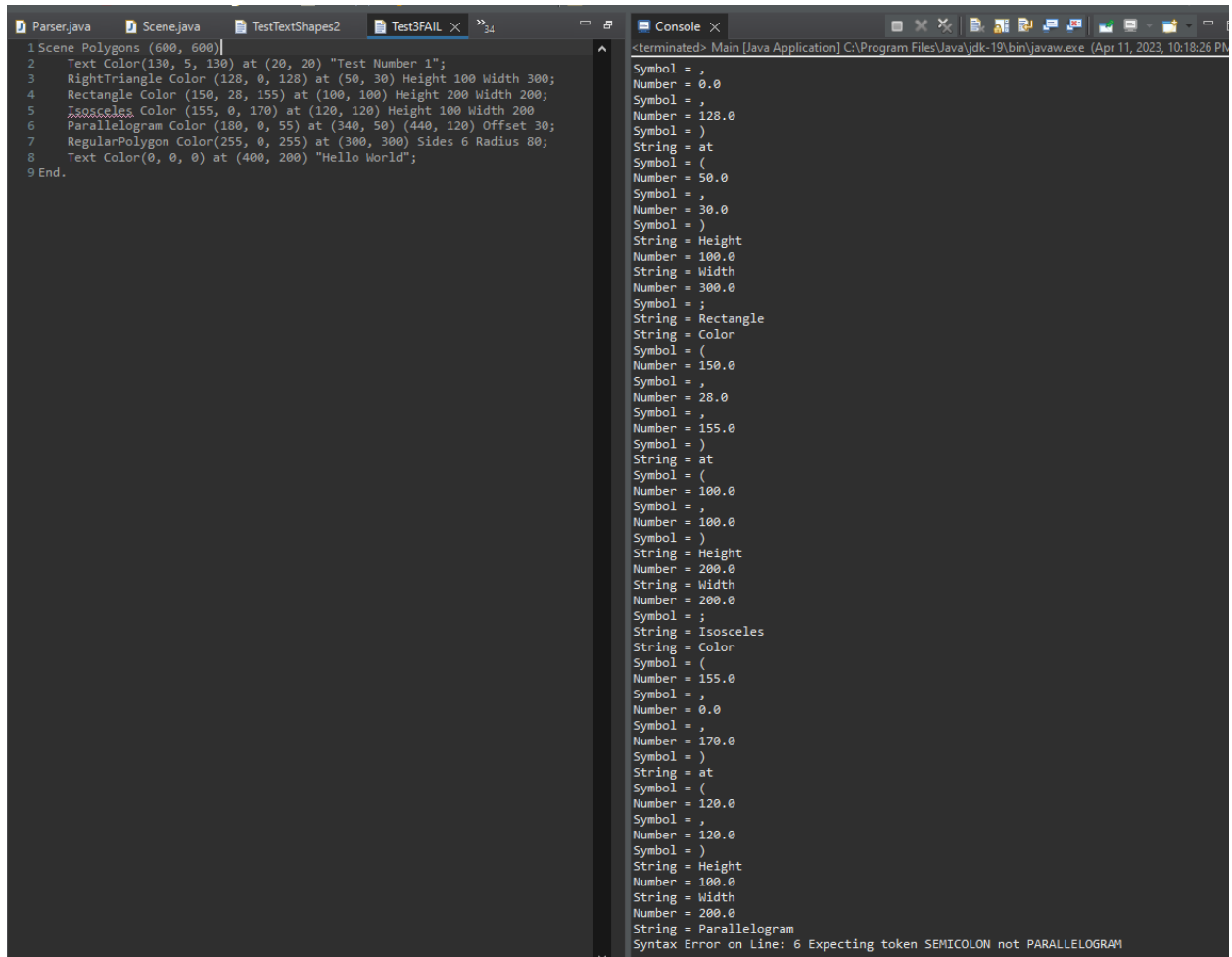


The Second Test I wanted to make the Rectangle fit all the other shapes inside and change the colors to represent United States Flag (red, white and blue), it took me a while to fiddle around with the x and y axis numbers, but eventually I was able to do it.



My Third and final Test was to purposefully make sure the compiler threw an error. I left out a Semicolon on the fifth line, and the program properly caught it, and returned the error message

in the terminal.



The screenshot shows a Java IDE with two panes. The left pane displays the source code of a test file named 'Test3FAIL.java'. The code defines a scene with various geometric shapes and text, including a RightTriangle, Rectangle, Isosceles triangle, Parallelogram, and RegularPolygon. The right pane shows the console output, which lists the symbols and numbers encountered during parsing. A syntax error is reported at line 6: 'Syntax Error on Line: 6 Expecting token SEMICOLON not PARALLELOGRAM'.

```
1 Scene Polygons (600, 600)
2 Text Color(130, 5, 130) at (20, 20) "Test Number 1";
3 RightTriangle Color (128, 0, 128) at (50, 30) Height 100 Width 300;
4 Rectangle Color (150, 28, 155) at (100, 100) Height 200 Width 200;
5 Isosceles Color (155, 0, 170) at (120, 120) Height 100 Width 200
6 Parallelogram Color (180, 0, 55) at (340, 50) (440, 120) Offset 30;
7 RegularPolygon Color(255, 0, 255) at (300, 300) Sides 6 Radius 80;
8 Text Color(0, 0, 0) at (400, 200) "Hello World";
9 End.
```

```
<terminated> Main [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Apr 11, 2023, 10:18:26 PM)
Symbol = ,
Number = 0.0
Symbol = ,
Number = 128.0
Symbol = )
String = at
Symbol = (
Number = 50.0
Symbol = ,
Number = 30.0
Symbol = )
String = Height
Number = 100.0
String = Width
Number = 300.0
Symbol = ;
String = Rectangle
String = Color
Symbol = (
Number = 150.0
Symbol = ,
Number = 28.0
Symbol = ,
Number = 155.0
Symbol = )
String = at
Symbol = (
Number = 100.0
Symbol = ,
Number = 100.0
Symbol = )
String = Height
Number = 200.0
String = Width
Number = 200.0
Symbol = ;
String = Isosceles
String = Color
Symbol = (
Number = 155.0
Symbol = ,
Number = 0.0
Symbol = ,
Number = 170.0
Symbol = )
String = at
Symbol = (
Number = 120.0
Symbol = ,
Number = 120.0
Symbol = )
String = Height
Number = 100.0
String = Width
Number = 200.0
String = Parallelogram
Syntax Error on Line: 6 Expecting token SEMICOLON not PARALLELOGRAM
```

Test3Fail.png

Other than learning about Java's tools for Tokenizing, Parsing, Graphical Interface and the importance of Class Hierarchy, making sure the Input Files are accessible and Grammar rules are properly established within the test files, I am reminded of how important and powerful O.O.P is when working on a multifaceted program. Breaking things down into Classes and using Objects that can be passed between classes is very helpful for creating a successful and maintainable program. The only thing that confused me a little was that the skeleton code had a few comments that threw me off. I forget what it was, but I think the RegularPolygon class had a comment explaining it was a Triangle if I remember correctly. This is my third time taking this class, and I very much appreciate this new approach to trying to teach Advanced Programming. The Word Doc was a HUGE HELP, and showing the actual Code in advance was a blessing also. I very much enjoyed working on this project.

-Matthew

