HW1 – Strategizing about Shapes

Estimated time: 8-12 hours

Objectives

- Become familiar with using UML class diagrams to describe not trivial structures.
- Become familiar with the Strategy pattern
- Become familiar with basic unit testing techniques

Overview

In this assignment, you will look for opportunities to apply the strategy pattern while building a program that reads a set of shapes from a file, computes their areas, and writes out a summary of the total areas of all the shapes, plus a breakdown of that total area by shape type.

Instructions

Your program needs to work with a set of shapes of various types. Specifically, each shape can be a circle, ellipse, triangle (equilateral, isosceles, or scalene), square, rectangle, regular polygon, or convex polygon. If your memory of elementary geometry, take a few minutes to research the properties of this shapes online. From an object-oriented modeling perspective, here are some key thoughts to consider:

- All circles are ellipses where the two focal point are the same so the major and minor axis have the same length
- Some triangles, namely those that are equilateral, are regular polygons
- All squares are rectangles and regular polygons
- Not all rectangles are squares
- All rectangles are convex polygons

Your program must be able to read a set of shapes from at least two different kinds of input files: JSON or XML. Each kind of file should be able to hold an arbitrary set of shapes.

Once your program loads a set of shapes from an input file, it needs to compute the area of shape of each shape and then total up the areas by shape type.

Finally, it need to be able to either display summary of the totals to screen to write them to a CSV file. The summary should give the total area for all shapes and then a breakdown by type. For example, given an input file that only contain ellipses, squares, and non-equilateral triangles, a display to the screen might look something like the following:

```
Total area of all shapes:                    80
        Ellipses:                    20
```

Circles:		5	
Non-circle Ellipses:		15	
Convex Polygons:			60
Triangles:		30	
Isosceles:	20		
Scalene:	10		
Rectangles:		30	
Squares:	30		

An output CSV file might contain the following data, where the first column is a shape category, the second column is the parent category, the third column is the category name, and the four is the total area:

```
1,,Total area of all shapes,80
2,1,Ellipses,20
3,2,Circles,5
4,2,Non-circle Ellipses,15
5,1,Convex Polygons,60
6,5,Triangles,30
7,6,Isosceles,20
8,6,Scalene,10
9,5,Rectangles,30
10,9,Squares,30
```

Your program should accept input parameters from the user to determine the type of input, the input file, where the output should be presented, and the type of output.

Here is a list of steps:

1. Model the structural concepts involved in this system using UML class diagrams
2. Based on your model, create some sample data for valid instances of each kind of shape, in each of the file formats
3. Based on your model, think about invalid instances of each kind of shape and how your program should handle bad data that could lead to invalid shapes
4. Design your program, to
    a. Allow the user to specific the input file text, the input file, the output type, and the output distance (screen or file).
    b. Import valid instances of the various type kinds of shapes and safely discard invalid data
    c. Compute the areas for all the shapes in an input file
    d. Rewrite out a summary to the screen or a text file in the specified format.
5. Implement your program
6. Test the highest risk components of your program using executable unit test cases

Submission Instructions

Zip up your entire solution, including test cases and sample input files, in an archive file called CS5700_hw1_<*fullname*>.zip, where fullname is your first and last names. Then, submit the zip file to the Canvas system.

Grading Criteria

Criteria	Max Points
A clear and concise model of Shapes using UML Class Diagrams	20
A working implement, with good encapsulation, abstractions, inheritance, and aggregation, and appropriate use of the strategy pattern	40
Meaningful, executable unit test cases	30