Assign-08

Shapes – Adding Templates and Exceptions

This set of assignments (A-04, A-07 and A-08) are meant to give you practice (once again) at developing class definitions from scratch – but in this final installment – you will be adding an exception handling strategy as well as templates to your existing classes. The set of activities are broken down into a 3 stages.

- A-04 : Gets you to develop 3 classes (with inheritance and polymorphism) as well as a test harness
- A-07 : Gets you to use some operator overloading
- A-08 : Gets you to develop some template functions and exception handling

Commenting in Assign-04, Assign-07 and Assign-08

In order to give you practice creating and writing DOxygen style comments – I want you to comment these 3 assignments using DOxygen. This means that all of your class header and method header comments must be written so that DOxygen can extract them and produce the set of online documentation. As well, I want you to also have commented all of the different class' data-members and constant values (if any) to also be extracted by DOxygen. You will want to revisit the DOxygen-Dog example from Module-06 to remind yourself how this is done. One final DOxygen expectation … I want you to also have a main project page for the **Shapes** project. You can place whatever you want on this main project page, but remember that the purpose of this page is to inform the reader about the project – what is the project all about? What is it modelling? Perhaps you could also tell the reader about what underlying OOP concepts and techniques are being used in the implementation?

STEP 1 – Connecting to your Source Repository and Extracting your Starting Point

1. Since this exercise is based upon SM-03 – all you need to do is to sign-out your existing Shape code from your repository
    a. Create a directory on the local machine – e.g. C:\SETRepo
    b. Connect to your repository *in the cloud* (remembering your repository's URL and your login credentials) using the Tortoise SVN client and perform an *SVN Checkout.* What you should notice is that you have signed out your DisneyCharacter solution as well as your Shapes solution.
2. If you like you can erase the DisneyCharacter subdirectory as you won't be modifying that source in this exercise
3. Once you have the Shapes solution – you are ready to begin this exercise … so let's get into the requirements …

STEP 2 – The Requirements and Programming

In this exercise, you will take your existing classes from A-07 and add to them. Also in this stage you will create a new class called **CircleSquare**. An object of this type will be created whenever a Circle shape is added to a Square shape (or vice-versa) … if the conditions are right !! There are 2 possible renderings of a CircleSquare object – depending on how they are created:

- You could have an instance of this object named "Circle-Square" which is created when a square object is being added to a circle object (see Figure 1)
- You could have an instance of this object named "Square-Circle" which is created when a circle object is being added to a square object (see Figure 2)

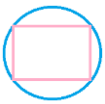

Figure 1

Figure 2

The Shape Class

- The Shape class should now also accept the names "**Circle-Square**" and "**Square-Circle**" as 2 more allowed shape names

The CircleSquare Class

- … Remember that this class is a child of the Shape class
- The class has the following data members
    - sideLength
        - a float data-type used to hold the side-length value (in centimeters) for the square
        - allowed values are greater than or equal to 0.00
    - radius
        - a float data-type used to hold the circle's radius value (in centimeters)
        - allowed values are greater than or equal to 0.00
- The class should have the following methods
    - SPECIAL NOTE

- When a circle-square object is being instantiated where a circle is being added to square – the name of the resultant object is "Square-Circle"
- When a circle-square object is instantiated where a square is added to a circle – the name of the resultant object is "Circle-Square"

- a constructor which takes values for the colour of the circle-square, its sideLength, the radius of the circle as well as the object's name / type
  - need to ensure that the sideLength value is valid …
  - need to ensure that the sideLength value is greater than or equal to the diameter (twice the radius) of the circle – otherwise, set the sideLength to 1.5 times the diameter of the circle
  - need to ensure that the name is one of Circle-Square or Square-Circle
  - no other input validation is necessary in this constructor
- a default constructor
  - sets the sideLength to a value of 0.00, the radius to a value of 0.00 and the name to "Circle-Square" and the colour to "undefined"
- a destructor that states "Closing the Circle-Square Ranch …"
- accessor for the data members
- mutator for the data members
  - your mutators do not need to pass back a succeed / fail status
  - but they do need to validate the input to ensure that it is proper for that data member and if it is not, then leave the attribute as it was
  - also remember about the special validation between the sideLength and radius of the circle ☺
- since the CircleSquare inherits from Shape, it will need to implement the Perimeter() and Area() methods
  - if the object is a Circle-Square – then the formula you implement is for the circle
  - if the object is a Square-Circle – then the formula you implement is for the square
- you will also need to implement the OverallDimension() method in this class
  - as you will notice from the sample output below – the CircleSquare class is special
  - depending on how it is created, it is either called a Circle-Square or a Square-Circle. The first shape in its name is the primary shape while the second part of its name indicates the secondary shape
  - in implementing the OverallDimension() method – please only implement it for the primary shape
- a method called `Show(void)` which prints out the shape's name, colour, radius, perimeter and area as in the following examples :

```
Shape Information
Name            : Circle-Square
Colour          : blue
Circle
  Radius        : 7.50 cm
  Circumference : 47.12 cm
  Area          : 176.71 square cm
Contained Square
  Side-Length   : 10.50 cm
  Perimeter     : 42.00 cm
  Area          : 110.25 square cm

Shape Information
Name            : Square-Circle
Colour          : orange
Square
  Side-Length   : 10.50 cm
  Perimeter     : 42.00 cm
  Area          : 110.25 square cm
Contained Circle
  Radius        : 3.75 cm
  Circumference : 23.56 cm
  Area          : 44.18 square cm
```

  - Note that part of the CircleSquare's output is to also show the perimeter and area for the *contained shape* (i.e. the secondary shape of the object)
    - How can you do this you ask ? Easy ! You'll need to know what type (CircleSquare or SquareCiricle) object you are dealing with to know what the "contained" shape is

Assign-08

Shapes – Adding Templates and Exceptions

        - CircleSquare contains a square
        - SquareCircle contains a circle
    - And you'll need to implement a method to determine the Perimeter() and Area() of the contained shape
- Add the following methods to this class – watch *best practices* !!
    - An overloaded + operation
        - When adding one circle-square to another, the resultant circle-square will take on the colour of the left-hand operand (the LHS)
        - The resultant radius and sideLength are the sum of the left and right operand's radii and sideLengths
    - An overloaded = operation – again following *best practices*

The Circle Class

- No changes

The Square Class

- Add the following methods to this class – watch *best practices* !!
    - Another overloaded + operation
        - Which will be called when the left-hand operand is a square, and the right-hand operand is a circle – this overloaded operator produces a circle-square object (the type is *Square-Circle*)
            - This means that the Square class needs to know about the Circle class and the CircleSquare class … hmmm …
        - When adding a circle to a square, the resultant circle-square will take on the colour of the square
        - The resultant circle-square object will have its sideLength be that of the square (the LHS) and its radius be that of the circle (the RHS)
        - It is important that this overloaded + operator check to make sure that sideLength is greater than or equal to the **diameter** of the circle (remember that the diameter is twice the radius) – if it is not, then throw an exception (can just be a string if you like) from this overloaded operator

The myShape Main

- Modify your testHarness to create the following shapes
    - CircleSquare shape called "playARoundSquare" – instantiated with the default constructor of the CircleSquare class
- Add the following template functions to your myShape source module
    - `CombineShape()` which is capable of taking 2 objects of the same shape class and combining them (using the "+" operation) and returning the new resultant shape
    - `CombineDifferentShape()` which is capable of taking a square object, a circle object and produces a circle-square object (using the new "+" operation developed in the Square class)
- Change your previous code which added square1 to square2 to use the `CombineShape()` template function instead
    - Make sure that playASquare is set to the result from this template function call
- Add the following calls to the end of your program
    - Call the `CombineDifferentShape()` template with square2 and round1 objects in order to produce and set the playARoundSquare object values (i.e. we want `square2 + round1`)
        - If no exception resulted – then print out the particulars of playARoundSquare
        - If an exception occurred - then
            - print the exception message
    - Then call the `CombineDifferentShape()` template with square1 and round2 objects in order to produce and set the playARoundSquare object values again (i.e. we want `square1 + round2`)
        - If no exception resulted – then print out the particulars of playARoundSquare
        - If an exception occurred - then
            - print the exception message

Assign-08

Shapes – Adding Templates and Exceptions

STEP 3 – Putting your Code back into the Repository

When you have made these changes to your class definitions and mainline – you can simply save your VS Solution and commit your files.

- Start by saving all files in solution and existing Visual Studio.
- In the previous exercise – you were reminded of repository *best-practices*
  - Where each source file needs to be committed individually with its own custom / specific comment
  - So locate all of your source files (.H files and .CPP files) and commit them one-by-one
- After committing the source files – you return to the C:\SETRepo directory where you do a final commit on the Shapes folder to pick up any stray files that also need to be committed

What to Submit

Before submitting you're A-08 solution, please remember to run DOxygen and have it extract your project and class documentation. As we saw in the Module-06 example, this produces all of the necessary web pages in an HTML directory. After running DOxygen and producing the output, please ZIP up the HTML folder into an HTML.ZIP file.

Regardless of whether you choose to store your class definition in a repository, please ZIP up the 9 source files [Shape.h, Circle.h, Square.h, CircleSqaure.h, Shape.cpp, Circle.cpp, Square.cpp, CircleSquare.cpp and myShape.cpp] as well as the HTML.ZIP file (with your DOxygen comments) and submit this single submission ZIP file into the assignment drop-box by the deadline.