Assign-04

Shapes – Laying the Foundation

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 this time, developing class definitions that will be involved in an inheritance relationship. 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

These assignments will be written up assuming that you will be placing the evolving code of these 3 exercises in your personal repository. It is okay if you don't do this … it is just a chance for you to get some more practice using an RCS. Below are the steps that you need to complete for this exercise.

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. I also want you to make sure to extract DOxygen information about any private or static methods. 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 Checking in the Initial Solution

1. Follow the instructions in the Creating-A-RiouxSVN-Repository document as well as the Installing-Tortoise-SVN document in order to create and prepare your computer for connecting to your personal repository code space.
2. As instructed in the Installing-Tortoise-SVN document in eConestoga, create a local directory on your computer (or even one of the lab computers) and call it **C:\SETRepo**
3. Connect to your personal 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 from A-02.
4. Now start Visual Studio (VS) and create a solution / project called **Shapes** in the C:\SETRepo\ directory
5. Let's setup the skeleton of the source files for this project …
    - Add files called Shape.h, Circle.h and Square.h to the *Header Files* section in the VS Solution.
    - Add files called Shape.cpp, Circle.cpp and Square.cpp to the *Source Files* section in the VS Solution and
    - Also add a file called myShape.cpp to the Source Files
    - When you add these files, simply place a single one-line comment in the file, so that they are not empty files
6. The reason I wanted to ensure that the source files weren't empty is because I want to **add** and **commit** the files to my repository now.
    a. As with DisneyCharacter, you ***do not want to add all files*** that are generated by VS. This is because some of the files change each and every time you open the solution, or are different on different people's computers …
    b. So save your solution and exit Visual Studio. Then go to C:\SETRepo\Shapes and erase the DEBUG folder. Then go into the C:\SETRepo\Shapes\Shapes folder and erase that DEBUG folder. You do not need or want these 2 folders in your golden copy of the repository.
    c. As well, if you venture into the C:\SETRepo\Shapes\.vs folder (which is hidden) and drill further down into the "…Shapes\v15\" folder, you may very well see a folder titled **ipch**. Erase this folder – this is where VS hides the incremental pre-compiled header files … these files take up unnecessary space in your repository.
7. Now you are ready to **add** and **commit** your new VS Solution
    a. Go back to the C:\SETRepo\ folder and right-click on the top-level **Shapes** folder
        i. Choose the *Tortoise-SVN* option from the pop-up and the *Add* option from the sub-menu – follow the prompts
        ii. Right-click on the same folder again and select *SVN Commit* – remember when prompted for a commit comment – we want to follow best practices – so give it a comment … perhaps "initial commit of Shapes solution and project directory structure and files"
8. Now your VS Solution and Project are ready to go, you have skeleton source files and they have been added to your repository …
    a. You are ready to proceed to Step 2 below – the class definition requirements and coding …

Assign-04

Shapes – Laying the Foundation

STEP 2 – The Requirements and Programming

As is the case with anything in life, there are a couple of rules that we need to play by …

- It is ***imperative*** that you write your own code … that is, you don't cut and paste from any of my examples, any of your existing code or any code found elsewhere!
- Having said that – you can look at your notes and the lessons, you can look at how I might have done something within my code or how you might have done something within your code … but resist the temptation to cut and paste – I am trying to see what you can produce from scratch.
- For the Stage 1 deliverable, you will start with a blank (new) Visual Studio C++ project that has the following files :
  - *myShape.cpp* – the test harness where your main() function will be …
  - *Shape.h* and *Shape.cpp* – the 2 source files where you will place your code for the **Shape** class
  - *Circle.h* and *Circle.cpp* – where you will place your Circle class definition
  - *Square.h* and *Square.cpp* – where you will place your Square class definition

Sound simple enough? Let's get into the particulars … In this exercise, you will be developing a **Shape** class, a **Circle** class and a **Square** class. As you may have guessed, **Shape** is the parent of **Circle** and of **Square**. Here are the details of these 3 classes:

The Shape Class

- The class has the following data members
  - name
    - a string-type[1] data member large enough to hold a value of up to 50 characters
    - allowed names are "Circle", "Square" and "Unknown"
  - colour
    - a string-type[1] data member large enough to hold a value of up to 10 characters
    - allowed colours include "red", "green", "blue", "yellow", "purple", "pink", "orange" and "undefined"

- The class should have the following methods
  - a constructor which takes values for the 2 data members and sets them
  - a default constructor
    - sets the name to "Unknown" and the colour to "undefined"
  - accessors for each of the data members … watch the return data type – you don't want to allow anyone calling the accessor to be able to change the data member's value!!
    - HINT – you want to make sure that you return your data member values as constant
  - a mutator for each of 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
      - if it is not, then leave the attribute value as it was
  - 3 ***pure virtual functions*** called `Perimeter(void)`, `Area(void)` and `OverallDimension(void)`
  - all methods are to be placed in the .CPP file

The Circle Class

- … Remember that this class is a child of the Shape class and will inherit publicly from it
- The class has the following data members
  - 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
  - a constructor which takes values for the colour of the circle and its radius
    - need to ensure that the radius value is valid …
    - no other input validation is necessary in this constructor
  - a default constructor

[1] You can choose to use a char[] for this data member or a string object … either way, you still need to be able to limit the maximum length to the size needed …

        - sets the radius to a value of 0.00
    - a destructor that states "The circle is broken …"
    - accessor for the data member
    - mutator for the data member
        - 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
    - a method called `Show(void)` which prints out the shape's name, colour, radius, perimeter and area as follows

```
Shape Information
Name          : Circle
Colour        : red
Radius        : 3.56 cm
Circumference : 22.37 cm
Area          : 39.82 square cm
```

    - Use the following formulae for your `Perimeter(void)` and `Area(void)` methods
        - `perimeter = 2πr`
        - `area = πr²`
        - where π is 3.1415926 and r is the radius value
    - the `OverallDimension(void)` method should return the diameter of the circle (return the value `2r`)
    - all methods are to be placed in the .CPP file

The Square Class

- … Remember that this class is a child of the Shape class and will inherit publicly from it
- 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

- The class should have the following methods
    - a constructor which takes values for the colour of the square and its sideLength
        - need to ensure that the sideLength value is valid …
        - no other input validation is necessary in this constructor
    - a default constructor
        - sets the sideLength to a value of 0.00
    - a destructor that states "The square is squished …"
    - accessor for the data member
    - mutator for the data member
        - 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
    - a method called `Show(void)` which prints out the shape's name, colour, sideLength, perimeter and area as follows

```
Shape Information
Name          : Square
Colour        : blue
Side-Length   : 10.50 cm
Perimeter     : 42.00 cm
Area          : 110.25 square cm
```

    - Use the following formulae for your `Perimeter(void)` and `Area(void)` methods
        - `perimeter = 4s`
        - `area = s²`
        - where s is the sideLength value
    - the `OverallDimension(void)` method should return the side-length of the square (return the value `s`)
    - All methods are to be placed in the .CPP file

Assign-04

Shapes – Laying the Foundation

The TestHarness

- It is in this source file (myShape.cpp) that you place your `main()` function and drive all of the wonderful things that can be done with the various classes … effectively *testing* your classes.
- You can use the source file (SampleMain.zip) I've placed in eConestoga along with this exercise as a starting point … if you like – it has a couple of pre-written functions (C-Style) that you might want to use … or you can develop your own…
- In your mainline, I want you to simply create a circle object and a square object by
    - Asking the user for specifics about the shape you are trying to create
        - Ask for the shape's colour
        - Ask for the radius (in the case of the circle) and the side-length in the case of the square
    - As soon as you have this information from the user – you can instantiate the shape
        - Since you are getting user input ***before*** instantiating the objects, this means that you will need to dynamically instantiate the circle and square objects …
        - [HINT] I wish I ***new*** how to do this !
    - Once both shapes have been instantiated, simply print out the specifics of each shape to the screen (using the Show() method)
- This is the end of this programming exercise … make sure that your class definitions follow the requirements and that your testHarness works.

STEP 3 – Putting your Code back into the Repository

When you have written these class definitions and method bodies – you can simply save your VS Solution and commit your files.

- Start by saving all files in solution and existing Visual Studio.
- In the DisneyCharacter 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-04 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 7 source files [Shape.h, Circle.h, Square.h, Shape.cpp, Circle.cpp, Square.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.