# Assign-07

## *Shapes – Adding Operators*

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 second installment – you will be adding some overloaded operators 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 A-04 – 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-04 and add to them :

### *The Shape Class*

- No changes

### *The Circle Class*

- Add the following methods to this class – you can add to the .H file – watch *best practices* !!
    - An overloaded + operation
        - When adding one circle to another, the resultant circle will take on the colour of the left-hand operand (the LHS)
        - The resultant radius is the sum of the left and right operand's radii
    - An overloaded * operation
        - When multiplying one circle with another, the resultant circle will take on the colour of the right-hand operand (RHS)
        - The resultant radius is the product of the left and right operand's radii
    - An overloaded = operation
    - An overloaded == operation
        - Which will return that 2 circles are equal if the radius and colours match

### *The Square Class*

- Add the following methods to this class – you can add to the .H file – watch *best practices* !!
    - An overloaded + operation
        - When adding one square to another, the resultant square will take on the colour of the left-hand operand (the LHS)
        - The resultant sideLength is the sum of the left and right operand's sideLengths

# Assign-07

## *Shapes – Adding Operators*

- An overloaded * operation
  - When multiplying one square with another, the resultant square will take on the colour of the right-hand operand (RHS)
  - The resultant sideLength is the product of the left and right operand's sideLengths
- An overloaded = operation
- An overloaded == operation
  - Which will return that 2 squares are equal if the sideLengths and colours match

### *The myShape Main*

- Modify your testHarness to create the following shapes
  - you can simply instantiate the objects required – no need for user input …
  - Circle shape called "round1" with radius 5.5 cm and colour "red"
  - Circle shape called "round2" with radius 10.5 cm and colour "blue"
  - Circle shape called "playARound" – instantiated with the default constructor of the Circle class
  - Square shape called "square1" with sideLength of 5 cm and colour "orange"
  - Square shape called "square2" with sideLength of 12 cm and colour "purple"
  - Square shaped called "playASquare" – instantiated with the default constructor of the Square class
- After creating these shapes, call the appropriate method to print out the specifics of each shape
- Then use your overloaded operators to
  - Add round2 to round1 and store it in playAround (i.e. `round1 + round2`)
    - You should end up with playARound being "red" with radius of 16.0 cm
  - Add square1 to square2 and store in playASquare (i.e. `square2 + square1`)
    - You should end up with playASquare being "purple" with sideLength of 17 cm)
  - Print out the specifics of playARound and playASquare
  - Multiply round1 by round2 and store in playARound (i.e. `round1 * round2`)
    - You should end up with a playARound being "blue" with a radius of 57.75 cm
  - Multiply square2 by square1 and store in playASquare (i.e. `square2 * square1`)
    - You should end up with a playASquare being "orange" with a sideLength of 60 cm
  - Print out the specifics of playARound and playASquare
  - Assign round1 to playAround, and then test to see if they are equivalent
    - i.e. `playRound = round1;` followed by `if(playAround == round1)`
    - If they are – then print "Hurray !!" to the screen
    - If they are not equal – then print "Awww !!" to the screen

## 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-07 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.