HW #6 (Inheritance & Polymorphism)

- For this project, you will create a series of classes to represent some simple geometric shapes and a small program to test your classes.

1. **class Shape**: Shape is an **abstract base class**.

1a. Data members: A Shape has the following private data member: a **color**, which is of type **string**.

1b. Methods: The Shape class should have the following public methods:

- a constructor that takes a **const string&** argument and uses it to initialize the shape's color. Since the Shape class is abstract, this constructor will only be invoked by a derived-class constructor.

HW #6 (2)

- a **virtual** method called **print()** that takes no arguments and returns nothing. The method should print the **color**.
- a **pure virtual method** called **get_area()** that takes no arguments and returns a **double**. Since it is pure virtual (or abstract), this method has no definition, only a prototype. It must be defined in any concrete class derived from Shape.

2. **class Circle**: Circle is derived from Shape using **public** inheritance.

2a. Data members: A Circle has the following **private** data member: a **radius**, which is of type **int**.

HW #6 (3)

2b. Methods: The Circle class should have the following methods:

- a constructor that takes a **string** to initialize the circle's color and an **int** to initialize the circle's radius. The color string should be passed to the Shape constructor.
- an overridden version of **print()** that takes no arguments and returns nothing. The method should call the base class **print()** method to print the color, then print the word "circle" followed by the circle's radius and area, e.g.:

  green circle, radius 10, area 314.159

HW #6 (4)

- an overridden version of **get_area()** that takes no arguments and returns a **double**. This method should compute and return the circle's area based on its radius.

3. **class Rectangle**: Rectangle is derived from Shape using **public** inheritance.

3a. Data members: A Rectangle has the following **private** data members:

- a **height**, which is of type **int**.
- a **width**, which is of type **int**.

HW #6 (5)

3b. Methods: The Rectangle class should have the following methods:

- a constructor that takes a **string** to initialize the rectangle's color and two **int**s to initialize the rectangle's height and width. The color string should be passed to the Shape constructor.
- an overridden version of **print()** that takes no arguments and returns nothing. The method should call the base class **print()** method to print the color, then print the word "rectangle" followed by the rectangle's height, width, and area, e.g.:

  red rectangle, height 8, width 6, area 48

HW #6 (6)

- an overridden version of **get_area()** that takes no arguments and returns a **double**. This method should compute and return the rectangle's area based on its height and width.

4. **class Triangle**: Triangle is derived from Shape using **public** inheritance.

4a. Data members: A Triangle has the following **private** data members:

- a height, which is of type int.
- a base, which is of type int.

HW #6 (7)

4b. Methods: The Triangle class should have the following methods:

- a constructor that takes a **string** to initialize the triangle's color and two **int**s to initialize the triangle's height and base. The color string should be passed to the Shape constructor.
- an overridden version of **print()** that takes no arguments and returns nothing. The method should call the base class **print()** method to print the color, then print the word "triangle" followed by the triangle's height, base, and area, e.g.:

    black triangle, height 4, base 10, area 20

HW #6 (8)

- an overridden version of **get_area()** that takes no arguments and returns a **double**. This method should compute and return the triangle's area based on its height and base.

Main program:

- Write a test program that creates an STL **vector** of pointers to Shape objects.
- Dynamically create some Circles, Rectangles, and Triangles (at least **two** of each). After creating each object, add it to the vector.

HW #6 (9)

- Loop through the vector of Shape pointers and call the **print()** method for each of them.
- Loop through the vector of Shape pointers again and call the **print()** method for each of the **Circle** objects in the vector.
- Loop through the list of Shape pointers one more time and delete each object.

- Output from this program should look something like this:

HW #6 (10)

Printing all shapes...

green circle, radius 10, area 314.159
red rectangle, height 8, width 6, area 48
yellow triangle, height 8, base 4, area 16
black triangle, height 4, base 10, area 20
orange circle, radius 5, area 78.5397
blue rectangle, height 3, width 7, area 21

Printing only circles...

green circle, radius 10, area 314.159
orange circle, radius 5, area 78.5397