Assignment 5 - Inheritance - Shape hierarchy

Necessary skills: Class inheritance, Member function overriding (virtual functions), Dynamic binding, Multiple classes (separate source and header files), C++ Standard Library string, iostream, iomanip, stringstream classes

Description

In this assignment, you'll create a hierarchy of C++ classes representing various shapes (eg. circle, rectangle, square). Each class will have its own `.cpp` source file and `.h` header file..

For the Shape class hierarchy, create an abstract Shape class that will be the base class of all the other shapes. It should have one private std::string instance variable representing the shape's color, and should have the following public member functions:

- *Shape(const string& color)* - a constructor that sets the color instance value.
- *string getColor()* - a const member function returning the object's color value.
- *double area()* - a const pure virtual member function that computes and returns the object's area. It must be overridden in each derived class.
- *string toString()* - a const pure virtual member function that returns the shape's description (color, type, measurements, and area) as a std::string. It must be overridden in each derived class. See the example output below for the format.

Create a `Circle` class that is derived from `Shape`. It should have one private `double` instance variable representing the radius, and should have the following public member functions:

- *Circle(const string& color, double radius)* - a constructor that invokes the base `Shape` constructor (passing the color), then sets it's own radius instance value.
- *double area()* - this overriding member function computes and returns the Circle object's area value.
- *string toString()* - this overriding member function returns the Circle object's description (color, type, measurements, and area).

Create a `Square` class with one private `double` instance variable representing the side length, with a constructor and overriding member functions for `area()` and `toString()`.

Create a `Rectangle` class with `double` instance variables representing the width and length, with a constructor and overriding

member functions for `area()` and `toString()`.

Each derived class .h header file should #include the base `Shape.h` header. Each derived class .cpp source file should #include its header (e.g. Circle.cpp has a #include Circle.h), plus the other header files needed (eg. iomanip, strstream)

Each derived class constructor must use the constructor initializer syntax to call the base `Shape` constructor with the color parameter.

Derived classes do *not* provide a member function override for `getColor()`. The base `Shape` class implementation is inherited.

All character data is held using std::string from the C++ Standard Library (no char* or char arrays).

All toString() methods use a local std::ostringstream object to format the output (using manipulators) using output stream syntax (<< insertion operators), and return the string contents of the ostringstream. These methods do not write any output, just return the formatted string (as shown below in the example) - the main program does the actual output of the string. The toString() methods will call their area() methods to obtain the area value to be formatted.

getShape.cpp

For reading shape data, create a getShape.cpp file containing a getShape() function that returns a base Shape * pointer. It should read a shape description from an input stream, create the correct type of derived shape with the `new` operator and parameters to the constructor, and return a base Shape * pointer to the new object. After reading a shape color and type (e.g. blue circle), it reads the additional information specific to that type of shape (e.g. for a circle, it reads the radius), and then uses the `new` operator to create the specific derived type of shape (e.g. new Circle(color, radius))

Input can be from cin, or from a stream input file (use iostreams). When there is no more input data (or when 'done' is entered), return a NULL Shape pointer. All character data is kept in C++ strings (no char[] arrays).

This is the only file that #includes the Circle.h, Square.h, and Rectangle.h header files (since it needs them to create the various types of shapes with the `new` operator).

main.cpp

The main.cpp file contains the main() function. It defines an array of base Shape pointers, and loops calling getShape() and storing the returned Shape pointer into the next available element of the array. The loop completes when a NULL Shape pointer is returned from getShape.

Note that the array contains pointers that point to a variety of derived shape objects (circles, squares, rectangles). This is possible because of the IS-A public inheritance relationship (e.g. a Circle IS-A Shape, and thus has all the properties and behaviors of a Shape).

Once all the shapes have been read, main() then loops, printing the list of shapes by calling the toString() member function on each Shape* pointer in the array.

main() next sorts the array of Shape pointers into ascending order by area. Note that the areas of the shapes need to be compared, but the array to be sorted contains Shape pointers. When the sort is complete, main then loops and prints the sorted list of shapes.

Finally, main loops and calls `delete` on the Shape pointers to delete the shape objects.

main.cpp() #includes *only* the base Shape.h header file (and the string, iostream, etc., headers needed for stream output). It does not (directly or indirectly) include Circle.h, Square.h, or Rectangle.h. It only needs the base Shape.h header to use the member functions defined in class Shape, but implemented (differently) in all classes derived from Shape.

Note that the `main()` method is an example of *generic object oriented programming*. It fills an array of pointers to various types of shapes, prints them, sorts them, and prints them again, without knowing the exact types of the Shape objects it is working with (and doesn't need to)! This is a very common and powerful OOP technique. All objects derived from Shape *implement the same set of inherited member functions*, and thus code using them can be written to generically work with Shape objects.

Sample output `(using interactive input)`

```
Enter a list of shapes - 'done' to end

Enter the shape's color (or 'done')...
red
Enter shape type...
circle
Enter the radius...
5.0

Enter the shape's color (or 'done')...
green
```

```
Enter shape type...
rectangle
Enter the length and width...
2.0 4.0

Enter the shape's color (or 'done')...
blue
Enter shape type...
square
Enter the length of a side...
3.0

Enter the shape's color (or 'done')...
done

The list of shapes entered...
  red circle with radius of 5.00 and area of 78.54
  green Rectangle with length of 2.00 and width of 4.00 and area of 8.00
  blue Square with side length of 3.00 and area of 9.00

Sorting shapes into order by area...
Sorting completed - 2 sort passes required

The sorted list of shapes...
  green Rectangle with length of 2.00 and width of 4.00 and area of 8.00
  blue Square with side length of 3.00 and area of 9.00
  red circle with radius of 5.00 and area of 78.54
```

Test Data

Use the following test data values.

```
red circle, radius 3.0
blue square, side length 4.0
green rectangle, length 3.0, width 2.0
blue circle, radius 2.0
```

```
red rectangle, length 8.0, width 2.0
green circle, radius 1.0
red square, side length 2.0
```

Formatted for a data file:

```
red circle 3.0
blue square 4.0
green rectangle 3.0 2.0
blue circle 2.0
red rectangle 8.0 2.0
green circle 1.0
red square 2.0
```