Lab 8: Inheritance & Polymorphism

CPSC 1021

School of COMPUTING Clemson University

Lab Objective

- Use an abstract base class
- Use inheritance to create a class from an existing class
- Use virtual functions and function overriding to achieve runtime polymorphism
- Practice solving programming problems using OOP concepts and techniques
- Practice incremental program development

Collaboration Policy

Work with your assigned team members from Lab 0. ***Please make sure to integrate your team members who – because of COVID requirements – have to participate remotely.*** You can use the MS Teams video conferencing feature to collaborate with a remote team member. Use the search bar on the top of the MS Teams app to search for your team member with their CU user name, then click the video icon in the top-right corner of the screen to start a video call.

You and your partners can work on the assignments together.

Submission Policy

Code solutions to all challenges **can be submitted as teams** this week. This means only one member from each team needs to submit the code files on Gradescope (this person must then add all their other team members to the submission). However, **you are required to work on all problems together**.

Introduction

In this lab, we will revisit the area calculation problem from lab 1 and lab 3, this time using inheritance and polymorphism to solve the problem. For the following exercise, you will implement an abstract base class `Shape` with a pure virtual function `getArea()`. You will use this abstract class to inherit the four shape classes you implemented in lab 3: circle, square, rectangle, and trapezoid. Each of these classes will have its own implementation of the `getArea()` function. Finally, you will write a program that uses (runtime) polymorphism to print the name and area of each shape by calling the respective functions from the base class.

Template Files

For this exercise, we have prepared a complete set of template files (uploaded to Canvas), which includes a main.cpp file and a Makefile. While most of the code is missing from these files (your job is to implement the missing code), the program can already be compiled with 'make'. Start with these template files and **incrementally** add code; frequently compile your program with 'make' to check that your code compiles without errors or warnings.

Area Calculation using Inheritance and Polymorphism

Below are the UML Class Diagrams for the five classes you need to implement.

Shape	
- name: string	
+ Shape() + Shape(name: string) + printName(): string + getArea(): double	 [mark as final] [pure virtual/abstract function]

Circle	
- radius: double - PI: double, const	 [set to 3.14]
+ Circle() + Circle(radius: double) + getArea(): double	

Square
- side: double
+ Square() + Square(side: double) + getArea(): double

Rectangle
- length: double - width: double
+ Rectangle() + Rectangle(length: double, width: double) + getArea(): double

Trapezoid
- base1: double - base2: double - height: double
+ Trapezoid() + Trapezoid(base1: double, base2: double, height: double) + getArea(): double

1. Start by implementing the abstract base class **`Shape`**. This class provides the interface for the more specialized class:
   - a function **`printName()`** that returns the string stored in the private variable 'name'. This function must be marked 'final' to prevent overriding.
   - an abstract function **`getArea()`** that will be implemented differently in each derived class.

   (Make sure your program still compiles after you have implemented this class!)

2. Next, implement the four specialized classes **`Circle`**, **`Square`**, **`Rectangle`**, and **`Trapezoid`**. Each class must be inherited from the abstract base class and add the following members:
   - private member variables that define the shape
   - a default constructor
   - an argument constructor that:
     - uses member list initialization to set the values of the private variables
     - sets the inherited variable 'name' to the name of the specialized shape. These names are hard-coded in the constructor as either "Circle", "Square", "Rectangle" or "Trapezoid". For example, constructor **`Circle()`** sets name to "Circle".

   (Make sure your program still compiles after you have implemented each class!)

3. Finally, edit the main.cpp file. This file must include:
   - four objects named **`circle`**, **`square`**, **`rectangle`**, and **`trapezoid`** instantiated from the four specialized classes. Initialize each object to the values given in the template file.
   - a vector implemented as follows:

     `vector<Type>shapeVect {element0, element1, element2, element3}`

     where 'Type' is a pointer to class **`Shape`** and each element is the address to one of the objects you have instantiated
   - a range-based for loop that iterates through **`shapeVect`** and prints the name and area of each object using functions **`printName()`** and **`getArea()`** accessed through the base class pointer.

   (Make sure your program still compiles after you have edited main.cpp!)

When you are finished, run your program with ‘./main.out’ or ‘make run’. Your program must produce the following output:

```
Area of Circle is: 12.6
Area of Square is: 4.0
Area of Rectangle is: 7.5
Area of Trapezoid is: 12.0
```

Submission Instructions

- Submit your complete and correct program to Gradescope. Your submission must include the following 11 files:
  1. main.cpp
  2. Shape.h
  3. Circle.h
  4. Circle.cpp
  5. Square.h
  6. Square.cpp
  7. Rectangle.h
  8. Rectangle.cpp
  9. Trapezoid.h
  10. Trapezoid.cpp
  11. Makefile
- Make sure your program compiles and runs on the SoC Linux machines; the autograder automatically assigns 0 points to programs that do not compile.
- Code style (proper indentation, consistency, readability, use of comments, etc.) will be considered when grading your submission.