Lab 11: Inheritance in Object Oriented Design

CPSC 1021 – Spring 2020

CLEMSON School of COMPUTING

Due Date: Friday, April 10, 2020 before 5pm

Lab Objective

- Implement a simple inheritance relationship
- Override methods

Introduction

At this point in the semester, we have covered many C++ language features that will come useful during this lab: STL containers, iterators, and of course classes.

The common thread in advanced C++ features is the idea that our programs should abstract away the bare-metal details that we had to worry about in C. When designing complex systems, we don't always want to concern ourselves with 1's and 0's, memory addresses, and counters. Instead, we need to focus on the high-level behavior of our system and make sure it accomplishes all the goals we set out for it.

Now we have learned about inheritance, which is simply a way to associate classes and share functionality. In this lab we will create a zoo of Animals, implement simple inheritance relationship, and override methods of the parent class.

Resources

http://www.cplusplus.com/doc/tutorial/inheritance/

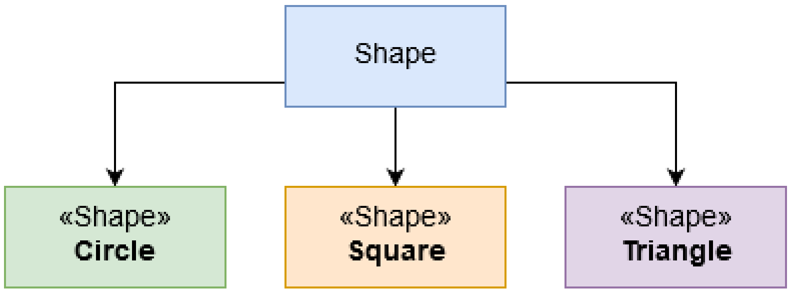
Part I - Inheritance

Inheritance

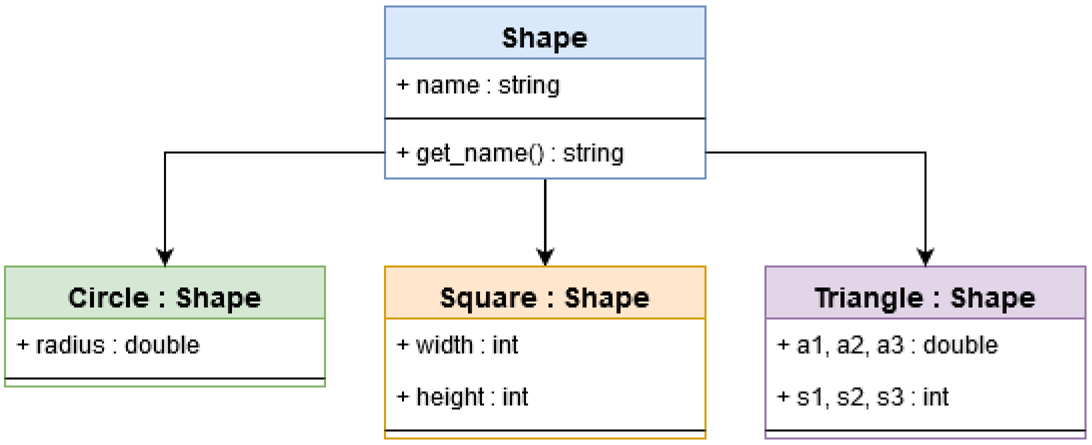
The most common metaphor used to explain inheritance is that of a parent and child. This can certainly be an accurate description of how inheritance relationships work, but real software often uses much more complicated relationships than simple parent-child. Let's take a brief look at this simple inheritance relationship.

Parent-Child Relationship

We use the familial vocabulary of parent-child when describing inheritance, because it is a relationship we are all familiar with. A child resembles and has all the traits of its parent and has new traits or features that make it unique. Consider the following class diagram:



This type of a relationship is a classic parent-child achitercture. The Circle, Square, and Triangle classes are children of the parent Shape class. We would often say that a Circle is a Shape, or that a Triangle is a shape. All the child classes have the same shared features as the Shape class while having unique properties of their own. For example:



Notice the unique properties of each child class: Circles have a radius, Squares have a width and a height, etc. Each of the child classes has access to the data and functionality of the Shape class itself. Study the following piece of code.

```
Shape s(“shape1”);
Circle c(“circle1”, 0.5);
. . .
s.get_name();  // OK => shape1
c.get_name();  // OK => circle1

s.get_radius(); // FAIL => No known method for class Shape
c.get_radiu();  // OK => 0.5
```

Circles know how to access both a radius and a name, while Shapes only have access to a name field. This is a straightforward and simple implementation of inheritance: when our classes move from general to specific.

Part II. Creating a Zoo.

In this assignment you will create a Zoo with two types of Animals.

1. Create class Animal that lives in *animal.h* and *animal.cpp* files. This will be the parent class of two other animals. The UML model is shown below.

Animal
-age: int -weight: double
+Animal (int, double) +getAge (): int +getWeight (): double +setAge (int): void +setWeight (double): void +printInfo(): void

2. Now create class Monkey and class Tiger. Each will live in their own .h and .cpp file. Both will inherit from class Animal. Please note that Monkey sounds like "Screeeech!!!", and Tiger sounds like "Roar!" (at least for the purpose of this lab).

Monkey
-name: string -soundsLike: string
+Monkey (string, string) +getVoice(): string +getName(): string +printInfo(): void Add setters as well.

Tiger
-name: string -soundsLike: string
+Tiger(string, string) +getVoice(): string +getName(): string +printInfo(): void Add setters as well.

3. Now create a driver named zoo.cpp. It will contain your main method. In the main method please do the following:

   a. Create two Tiger objects. Instantiate them using some data of your choice. Add them to the vector of Animals called myAnimals.

   b. Create two Monkey objects. Initialize them with data of your choice. Add them to the same vector.

4. Write an implementation for the method printInfo in Animal, and then override that method in each of the derived classes. Information printed by that method has to be specific to that class.

5. After you create the zoo and all the animals, use an iterator to iterate through the vector and print all animals with their data in some organized manner. You also have to demonstrate that all your methods work.

6. As usually, please create a Makefile.

What/How to submit

That's it, you are done! You can now submit your zipped tarball to canvas to be graded.

NOTE: If you submit the archive that cannot be open, or is corrupt, you will not be given the opportunity to redo the work and resubmit. You have one shot to get it right.

Congratulation!! the labs are now finished. This was your last lab in this course. You have learned a lot of advanced C++ features that will be very useful in your higher level computer science courses.

How lab grade is calculated.

Since we had 11 labs, not 12, as was projected at the beginning of the semester, the total number of points that could be earned is 55 (11*5 = 55), not 60. This will not affect your grade in any way. Your lab grade is the number of points you earned out of 55. Attendance does not count towards your lab grade, even though occasionally TAs use gradebook attendance column to take attendance. This lab grade will be included by your lecture section teachers in the calculation of your final course, according to their section syllabus. Please contact your lecture section instructor, if you have questions about your course grade.