Lab 8: Inheritance

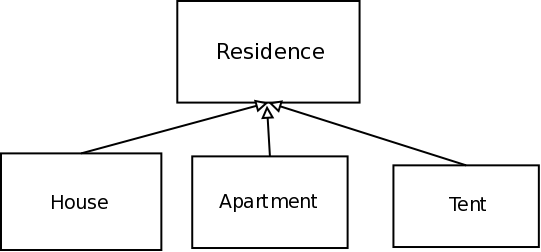
Objectives:

In this lab you will:

* Learn about **inheritance** in order to make more specialized derived classes that**extend** the functionality of a base class.
* Learn how to inherit instance variables and methods from the base class.
* Use **method overriding** to change the behavior of an inherited method when it is used by a more specialized derived class.
* Learn about the **protected** visibility modifier.

Description

Inheritance is used to implement an **"is a"** type of object relationship (for example, a Square **is a** type of Shape). In this lab, you have the following inheritance hierarchy:



The base class is called Residence and the derived classes are House, Apartment, andTent. The classes House, Apartment, and Tent inherit from the Residence class; thus they are more specialized versions of Residence.

The Residence class has private data members for storing:

* The number of rooms
* The number of walls
* A boolean for whether or not a Residence has a washer

Source Code Structure

Your program will consist of the following files:

1. [residence\_driver.cpp](http://www.csee.umbc.edu/courses/undergraduate/202/fall15_marron/labs/lab08/residence_driver.cpp): download this file into your lab08 directory. The file is also available on GL in the directory
2. /afs/umbc.edu/users/c/m/cmarron/pub/cmsc202

This will be your driver program. You will need to implement the classes and methods used by this file.

1. Residence.h and Residence.cpp
2. House.h and House.cpp
3. Apartment.h and Apartment.cpp
4. Tent.h and Tent.cpp

Residence Class

The constructor takes an integer for the number of rooms, an integer for the number of external walls, and a boolean to indicate whether or not there is a washer.

Residence also has four methods:

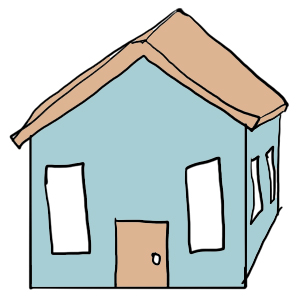
* HasWasher(), which returns a boolean. It returns true if there is a washer in the residence and false otherwise. This method should be **protected**.
* PropertyValue(), which returns a double. It calculates the property value based on number of rooms \* 10000.
* NumWindows(), which returns an int. It calculates the number of windows based on the number of external walls \* 2.
* operator<<, which returns an ostream& containing the number of rooms, the number of walls, whether or not there is a washer, the number of windows, and the property value. This will be a friend function, so it is not inherited.

The output of operator<< should be similar to the following.

|  |  |
| --- | --- |
| 1  2  3  4  5 | Number of Rooms: 4  Number of Walls: 4  Washer: true  Number of Windows: 8  Property Value: 40000.0 |

As a reminder, **public** members and methods are accessible from any class, while **private** members and methods are only accessible from within the class in which they are declared. The protected provides protection midway between public andprivate. Only classes inherit from a class with protected items may access those items directly.

House Class



This class inherits the instance variables and methods from Residence, so make sure you do the following:

|  |  |
| --- | --- |
| 1  2  3 | class House : public Residence {    }; |

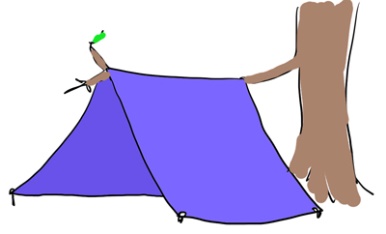
Since this is a subclass of Residence, you can call the constructors from Residence in your initilization list. The Houseconstructor should call it's parents constructor with the values 4, 4, and true.

Since House inherits everything from Residence, you don't need to write any new methods at all! House has aPropertyValue() and NumWindows() method just like Residence because it is a type of Residence.

So, when you want to know the property value of a House object h, you would call:

|  |
| --- |
| h.PropertyValue(); |

Tent Class



Overriding Methods

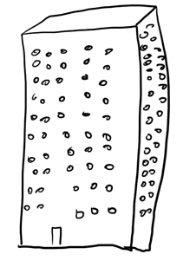
This class shows you how to override methods from a parent class.

Your tent will have 1 room, 2 walls, and will NOT have a washer.

Since a tent doesn't really have much value, it does not make sense to use the PropertyValue() method defined in Residence. Instead, you can override PropertyValue() by using the same method stub present in Residence, but changing the logic inside the method. So, you will make a method in Tent called PropertyValue() that returns a double, but all it does inside the method is return 0.

Also, a tent has no windows, so you should also override the NumWindows() method to return 0.

Apartment Class



(Overriding methods continued)

Your apartment will have 4 rooms, 4 walls, and may or may not have a washer.

This class builds on what you did in your House class. You will use the parent constructor in Apartment's constructor, but you also want to know which story your apartment is on, so you will give Apartment a private int instance variable for story. You will make int story a constructor input so you can set your Apartment's story when you create the object.

The property value now depends on whether or not the apartment has a washer. If the apartment has a washer, the property value goes up by 100. So, you will need to have the constructor also accept whether the apartment has a washer or not.

In addition to the more complex constructor and new instance variable, you will make a method to return the story on which your apartment exists.

You should also overload the insertion operator so that your Apartment now prints the property value.

Testing

We will be using the insertion operator to display the three objects created. Your output should look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | House:  Number of Rooms: 4  Number of Walls: 4  Washer: true  Number of Windows: 8  Property Value: 40000.0    Apartment:  Number of Rooms: 4  Number of Walls: 4  Washer: true  Number of Windows: 8  Property Value: 40100.0  Story: 2    Tent:  Number of Rooms: 1  Number of Walls: 2  Washer: false  Number of Windows: 0  Property Value: 0.0 |