COP3014: Project 1 – Classes and Top-Down Design

Overview

Software development projects using the object-oriented approach involve breaking the problem down into multiple classes that can be tied together into a single solution. In this project, you are given the task of writing some classes that would work together for providing a solution to a problem involving some basic computations.

Learning Objectives

The focus of this assignment is on the following learning objectives:

- Be able to identify the contents of class declaration header and class definition files and to segregate class functionality based on class description
- Be able to write object creation and initialization code in the form of constructors
- Be able to implement an encapsulated approach while dealing with data members for the classes involved through accessors and mutators (getters and setters)
- Be able to make the program code spread across multiple files work together for the solution to the computing problem

Prerequisites

- To complete this project, you need to make sure that you have the following:
- C++ and the g++ compiler
- A C++ IDE or text editor (multiple editors exist for developers)
- An understanding of the material presented in class
- An understanding of the material covered in zyBooks.

Problem Description

You are to implement the two classes as part of a used items store program that sells used furniture and used computers. The list of requirements are as follows:

General requirements for classes:

1. Each class should be declared in a file named as <**ClassName.h**> and implemented in a file named <**ClassName.cpp**> where ClassName is the name of the class. For example, a class named UsedFurnitureItem should be declared in a file named "**UsedFurnitureItem.h**" and implemented in a file named "**UsedFurnitureItem.cpp**".
2. Each class should provide a getter (accessor) and a setter (mutator) for each private member. <u>Setters should reject meaningless values such as negative age</u>.
3. The class should include a default constructor that initializes the private member variables to reasonable values.
4. The class should include an overloaded constructor that lets the initialization of the private member variables to values specified while creating an object of the class.

UsedFurnitureItem:

1. Create a class named ***UsedFurnitureItem*** to represent a used furniture item that the store sells.
2. Private data members of a ***UsedFurnitureItem*** are:
    a. **age** (double) // age in years – default value for
    b. **brandNewPrice** (double) // the original price of the item when it was brand new
    c. **description** (string) // a string description of the item
    d. **condition** (char) // condition of the item could be A, B, or C.
    e. **size** (double) // the size of the item in cubic inches.
    f. **weight** (double) // the weight of the item in pounds.
3. Private member functions of a ***UsedFurnitureItem*** object are
    a. double **CalculateCurrentPrice( ):** Current price depends on age, **brandNewPrice**, and condition of the used furniture Item. These three would be the arguments of this function. If the item is in A-condition, the current price will go down by extra 10% of the **brandNewPrice** for each year of its age until 7 years (e.g., After the first year, and item with a $100 **brandNewPrice** will cost $90, after 2 years, $ 80, and so on). If the age is greater than 7 years, the price is fixed at 30% of the **brandNewPrice**. The current price of B-condition items goes down by extra 15 % of the **brandNewPrice** for each year until the 5th year. After that, the current price is fixed at 20% of the **brandNewPrice**. Items with C-condition are priced at 10% of the **brandNewPrice** regardless of age.

    b. double **CalculateShippingCost( ):** Shipping cost of a **UsedFurnitureItem** depends on weight, size, and distance (arguments for the function). While weight and size are member variables, shipping distance is provided as an additional argument to this function. Shipping rate is 1 cent per mile for items that are smaller than 1000 cubic inches and smaller than 20 pounds. Items with larger size or weight cost 2 cents per mile.

4. Public member functions of a ***UsedFurnitureItem***
    a. Constructors – default and overloaded [default values: age = 1.0, brandNewPrice = 10.00, description = "Not available", condition = 'A', size = 1.0, weight = 1.0]
    b. Accessors (getters)
    c. Mutators (setters) – You should ensure that invalid (zero or negative) values do not get assigned to age, **brandNewPrice**, **size** or **weight** data members. An invalid character (other than 'A', 'B' or 'C') should not be allowed to be assigned to the **condition** data member.
    d. void **PrintInvoice( ):** This function calls the accessors to output the values of all data members and the current price and shipping cost by calling the **CalculateCurrentPrice( )** and **CalculateShippingCost( )** private functions.

UsedComputer:

1. Create a class named ***UsedComputer*** to represent a used computer item that the store sells as well.
2. Private data members of a ***UsedComputer*** are:
    a. **age** (double) // age in years
    b. **model** (string) // a string description of the computer
    c. **brandNewPrice** (double) // the original price of the computer when it was brand new
    d. **condition**: (char) // condition of the item could be A, B, or C.
3. Private member functions (can be called by another member function, but not freely from main() function) of a ***UsedComputer*** object are
    a. **CalculateCurrentPrice( ):** current price is calculated the same way the price for a *UsedFurnatureItem* object is calculated.

    b. **CalculateShippingCost( ):** shipping cost depends on distance which is provided as an argument. The shipping rate is 1 cent per mile.

4. Public member functions of a ***UsedComputer***
    a. Constructors – default and overloaded [default values: age = 1.0, brandNewPrice = 10.00, model = "Not available", condition = 'A']
    b. Accessors (getters)
    c. Mutators (setters) – You should ensure that invalid (zero or negative) values do not get assigned to **age**, or **brandNewPrice** data members. An invalid character (other than 'A', 'B' or 'C') should not be allowed to be assigned to the **condition** data member.
    d. void **PrintInvoice( ):** This function calls the accessors to output the values of all data members and the current price and shipping cost by calling the **CalculateCurrentPrice( )** and **CalculateShippingCost( )** private functions.

Test Program:

Develop a test program that will test all functionalities of these two classes.

a. The test program will create one ***UsedFurnitureItem*** and one ***UsedComputer*** object each by calling the default constructors.
b. The test program will then call the accessors for all data members of the objects created in step ***a***.
c. Next, the test program will create one ***UsedFurnitureItem*** and one ***UsedComputer*** object each by calling the overloaded constructors.
d. Next, the test program will test each setter (for objects of both types in a sequence) by calling the setter to set a value and then call the corresponding getter to print out the set value. This test should be done twice on data members that could be set to invalid values (that have numerical or character data type) – once after trying to set invalid values and subsequently, once after setting them to valid values. The data members with string data types (**model**, **description**) can be tested just once.
e. Finally, once all member variables are set, the private function **PrintInvoice( )** will be used to call the accessors to print the values of data members and the result of **CalculateCurrentPrice( )** and **CalculateShippingCost( )**.

Development Diary

For this project, you must complete the following tasks in the order that they are described:

1. Write down the list of all classes that you need to implement. For each class, list the associated member variables and the set of member functions.
2. Create a plan to implement your solution to the problem with a timeline. Each step in your plan refers to an increment of the program that you will be creating. It is recommended to complete the implementation of a single logical action per step.

Your responses to these tasks should be submitted as a PDF document called **lastname_firstname_project01.pdf.**

Grading Breakdown

- **[24 pts]** Full-coverage test driver. Driver should call all functionality and print out the state of the object after each change to the internal state. This test driver should print the expected value and the actual result.

- **[25 pts]** ***UsedFurnitureItem*** class fully implemented to the specifications provided in the problem description.
- **[25 pts]** ***UsedComputer*** class fully implemented to the specifications provided in the problem description.
- **[10 pts]** Sufficient and clear class divisions (with **main( )** responsible for I/O only)
- **[10 pts]** Clear, easy-to-read code (e.g. class declarations in .h files, class definitions in .cpp files, working makefile, etc.)
- **[6 pts]** Development "diary" (report) that details design and implementation of each class, as well as the project as a whole. Should include a reflection of the process as a whole upon completion of the project (identify the challenges, lessons learned, ways to improve in the future, etc.)

Submission
Points will be deducted for not following these instructions.

Before submitting this project in eLearning make sure that you follow the following steps:

1. Make sure that your name appears at the top of each file. This should be done as a comment for any source code files.
2. Put your development diary document (.pdf), source code (.h & .cpp), and working makefile into a .zip file with the file name "lastname_firstname_project01.zip".
3. Turn your **zipped-up** project into eLearning/Canvas.