CISP 400 - Lab Assignment #4
Due Thursday, September 27, 2018 at 11:59 PM

In this lab, we are going to put together a program that uses a relatively simple class to keep track of inventory management. It will include some code to demonstrate the function of objects of this class.

CLASS SPECIFICATION:

Design an Inventory class that can hold information for an item in a retail store's inventory.

The class should have three private member variables – itemNumber, of type int; quantity, of type int; and cost, of type double.

It should have three public member setter functions that set the private data, one for each variable: setItemNumber, setQuantity, and setCost. SetItemNumber is type void. setQuantity and setCost return bool, and they are detailed in the "Input Validation" section below.

Note that setItemNumber does NOT do validation at any point.

The Inventory class should have three public member getter functions that get the private data, one for each variable: getItemNumber, getQuantity, and getCost.

It should have one public member function, getTotalCost, that computes the total cost of the inventory on hand, and returns it. The calculation is the quantity times the cost.

The class should have two constructors. One should set all the member variables to zero. The other should have item number, quantity, and cost and parameters, which uses the setter member functions to put in the initial values. Note that it DOES NOT do input validation. Input validation in a constructor is beyond the scope of this assignment.

The Inventory class should have one destructor that announces that the object is being destroyed.

INPUT VALIDATION:

Two of the setter functions do more than simply set a variable.

setQuantity and setCost both return bool. They check to see if their parameter is negative (e.g. less than 0). Note that setQuantity takes an int parameter, because the quantity member variable is an int, and setCost takes a double parameter, because the cost member variable is a double.

If the parameter is negative, then the function sets the respective variable to 0 and returns false. If the parameter is not negative, then the function copies the parameter into the respective member variable and returns true.

DEMONSTRATION:

The rest of the program is going to be written to give a brief demonstration that this Inventory class actually functions as required.

This demonstration will use TWO inventory objects. One will be a pointer, one will be a local variable.

FIRST INVENTORY OBJECT:

The FIRST INVENTORY OBJECT will be dynamically allocated, meaning that it is going to use the Inventory type **pointer**.

We will gather an item number, quantity, and cost for this particular inventory item. These will then be used to allocate the inventory item, using the new keyword. Remember – use parameters to the new keyword, because otherwise the wrong constructor will be called.

This object does NOT have input validation used, so you can give it crazy values and it will not complain. You can have negative inventory or quantity with it with no problem.

Remember to ask the user for the variables in this order: item number, quantity, cost.

SECOND INVENTORY OBJECT:

The SECOND INVENTORY OBJECT will be a local variable. We are not going to pass along any specific parameters to its declaration so it will use the default constructor.

We will gather input – a SECOND TIME – from the user. This can be completely different from the first time data was input. In fact, it probably should be for proper testing. We will get an item number and IMMEDIATELY set it in the object, rather than simply holding onto it as we did with the first object.

We will then get a cost for the inventory item, and a quantity for the inventory item. But these are different. These will each be in a LOOP, so we can test out the validation capabilities of the setCost and setQuantity methods (remember – they're type bool).

Once we have the input, we will try to use the setter function for it. But notice the RETURN TYPE of the setter – it's bool, and we wrote it so it would check to see if the input is negative for the cost and quantity. Check this to see if the setter returned true or false. If it is true we can exit the loop. If it's false we tell the user they made a mistake and to enter again. The loop must repeat until a valid value has been given and then exit.

Both cost and quantity should be obtained in separate loops. So the program will continue to ask about one of them until it gets a valid value. It will then continue to ask the user about the other until it gets a valid value.

Remember to ask the user for the variables in this order: item number, quantity, cost.

SHOWING THE RESULTS:

We will write a function that accepts two Inventory objects by reference. REMEMBER – pass by reference – include the & character!

This function will call the print() method on each object and then exit.

Now, because it is pass by reference, we have one object we can use with it directly, the second object. The FIRST OBJECT is still handled through a pointer, so we will have to DEREFERENCE that pointer so it can be used in pass-by-reference. So the function call will look like:

printMyInventoryObjects(*firstObject, secondObject);

After all of this, the first object should be deallocated. We then exit. Each inventory instance should inform the user of its destruction by printing a message.

TIPS:

- cost and quantity are quite similar in terms of how to handle them, even though one is a double and the other is an int.
- Keep the member functions and class definition separate. Do not use inline functions.
- There is no validation checking for itemNumber – any int value will do!
- The operators for accessing members – the period (.) and the arrow (->) - are going to be tricky for this class. Remember to always use -> with pointers.
- Refer to how to call constructors so you can pass the parameters to each constructor effectively.
- You can call member functions within an object without any sort of prefix.

DELIVERABLES:

Upload the .cpp file to Canvas by the due date.