Store Backend - Part 2

Points

Points		Due
10	Design	March 31, 2017
90	Working Classes	April 5, 2017
100	**TOTAL**	

Submission

1. **Design**: Submit the PDF to eCampus.
2. **Source Code**: Submit the source code (**customer.cpp , customer.h , product.cpp, product.h, store.cpp and store.h** files) to Vocareum
   Note: you do not have to upload the file with the main() function. However, if you do, it will not affect grading. We will use one with our test cases when we compile the classes you provide.

Program

You will create a backend for organizing data for a store. The program will span three homeworks. All homeworks will be based on this UML Diagram. You will incrementally create the classes and indicated member functions each week. By the end, all classes and functions will be fully implemented.
This program will be tested using only your objects and their methods / member functions using a driver program created by the graders. However, you will need to create a "driver program" that you can run to develop and test your code. In the driver

program, you can set things up to test your objects. See the end of the page for some guidance on creating a driver program.

Specifications

Based on the UML Diagram, create the Store class and update the Customer, and Product classes.

- Include all attributes / data members as indicated in the UML Diagram.
- Implement the constructors and the methods / member functions listed below.
- *Note: Not all methods will be implemented in this homework. The remaining class and methods / member functions will be implemented in subsequent homeworks.*

Design

A lot of the design has been done for you in the UML Class Diagram provided.

1. Outline your steps for creating a driver program for testing the classes.
2. Create test cases to ensure the class behaves correctly when created and or updated.

Separate Files

Each class should be in a separate file with its own headerfile. By convention, the class name matches the name of the source (.cpp) and header (.h) files. **Do not forget to use header guards.**

const

Think about which member functions should not *change* the objects. Those member function declarations should be modified and marked as `const` for all classes.

Store Class

- `Store();`
- `Store(string name);`

- `string getName();`
- `void setName(string name);`
- `void addProduct(int productID, string productName);`
  Create a new Product and add it to products. If this productID already belongs to another product, throw an exception.
- `void addCustomer(int customerID, string customerName);`
  Create a new Customer and add it to customers. If this customerID already belongs to another customer, throw an exception.

Product Class Updates

Customer Class Updates

- `void processPayment(double amount);`
  Add amount to balance. If amount is negative, throw an exception.
- `void processPurchase(double amount, Product product);`
  - *If the customer has credit: subtract amount from balance. Recall that balance can be negative if credit is true.*
  - *If the customer does not have credit: if the balance is greater than or equal to the amount then subtract amount from balance. Otherwise throw an exception. Recall, balance is not allowed to be negative if credit is false.*
  - *If the purchase occurs, then add product to productsPurchased if it is not already there. If amount is negative, throw an exception.*
- `void listProductsPurchased(ostream& os);`
  Output information about each product purchased. While, not reflected in the UML, to allow for different output streams to function with it, we will pass in an ostream to use for the output. You should utilize the output operator for product class in this function.

Output Operators <<

Create overloaded output operators for the Product and Customer classes. Recall, these are helpers for those classes. So the function declaration should go in the header (.h) file for the corresponding class, but outside of the class definition. The function definition should go in the cpp file of the corresponding class.

Updated Driver Program Guidance

Your driver program will help you test and debug your objects as you create them. A nice thing about using a driver program, is you do not have to validate all of the inputs. You are only using it for testing, so you can ensure that the inputs that come in are valid. However, you still have to do validation that a member function is expected to do, e.g. when an exception is thrown.

Adding to the driver you started last time, you will probably want test some of the functions that interact with other classes. So part of your driver program might look something like this:

Main.cpp

```
#include <iostream>
#include "Store.h"
#include "Product.h"
#include "Customer.h"

int main() {
   // Create a Product
   Product p1(77, "Thing 1");
   Product p2(88, "Thing 2");
   Customer c("Bill Gates", 12345, true);
   c.processPurchase(250, p1);
```

```
        c.processPurchase(100, p2);
        c.listProductsPurchased(cout);
    }
```