COMP 2404 -- Assignment #1

Due: Thursday, February 2, 2017 at 9:00 PM

Collaboration: You may work in groups of no more than two (2) students; each student must contribute an **equal** amount of work

Goal

You will be working with an existing inventory management program throughout the term. The **base code** that you must start with is posted in *cuLearn*. For this assignment, you will modify the base code to add a new entity class and a new collection class. You will also implement a new feature and modify existing ones.

Learning Objectives

- understand an existing object-oriented program
- understand entity/control/UI classes, and their role in a program
- add code to a small program in C++

Instructions:

1. Draw the UML class diagram:

- Read and make sure that you thoroughly understand the existing inventory management program provided in the base code.
- Using a drawing package of your choice, draw a UML class diagram to represent the objects you identified. You must show all classes (except the array classes), their attributes and operations, the associations between the classes (inheritance or composition), and the multiplicity and directionality of all associations. Do not show getter and setter functions, nor constructors or destructors.
- You must update your UML diagram *after you have completed the rest of the assignment*. The UML diagram that you submit must match your code.

2. Implement the purchase and purchase collection classes

You will create a new purchase class (called `Purchase`) and a new purchase collection class (called `PurchArray)` to contain a collection of purchase objects. Each customer will have a purchase collection to track the products that they have bought, as well as the number of units of each product. Your classes will behave as follows:

- the `Purchase` class will contain the id of a purchased product, and the number of units of that product bought by that customer over the customer's lifetime
- the `PurchArray` class will contain an array of `Purchase` objects and its current number of elements
- you will modify the `Customer` class will contain a collection of purchases
- each customer's purchase collection will contain *only* products that the customer has actually bought; there should be no "placeholder" purchases and no purchases with duplicate product ids
- when a customer buys a product:
  - o if the customer has bought this product in the past, then it will already be in their purchase collection; you must find this product id in the purchase collection and increment the number of units bought
  - o if the customer has never bought this product before, then you must add the product as a new purchase in the purchase collection

3. Implement the "product purchase" feature

The "product purchase" feature allows a customer to buy several products. Buying a product earns the customer some loyalty points, which have a 1-to-1 correspondence with the product price (for example, if a product costs $5, the customer earns 5 points). The inventory management system tracks which products the customer buys and how many units of each product over time. This information is tracked with the purchase collection that you implemented in the previous step.

You will implement the "product purchase" feature that can be selected from the cashier menu. You will modify the `Customer` class to store the number of loyalty points that a customer has earned over their lifetime.

The "product purchase" feature will:

- prompt the user for the id of the customer making the purchase, and verify that it is an existing customer
- prompt the user for a collection of product ids that the customer wants to purchase, terminated by zero
- for each product purchased:
  - verify that the product exists
  - verify that there is at least one unit of this product in stock
  - reduce the number of units of that product available in the store
  - compute the number of loyalty points earned by the customer with purchasing this product
  - add the loyalty points to the customer's points
  - keep a tally of the total purchase amount (the product prices) and the total number of loyalty points
  - register the purchase in the customer's collection of purchases
    - if this is a repeat purchase, then simply increment the number of units bought
    - if this is a new purchase, then you must add the product as a new purchase
- print a summary to the screen of the total purchase amount and number of points earned

Notes:

- For *this assignment only*, if a customer or product cannot be found, the program will terminate
- Sloppy design will be penalized. The code for this feature belongs in several different classes, in keeping with correct design principles. You must think this through and plan where to place which parts of the code.

4. Modify the "print customers" feature

You will modify the "print customers" feature so that it prints out all the purchase information for each customer, in addition to all the customer's information.

Note:

You will provide sufficient *datafill* for products and customers. This means that your program must:

- create and initialize a minimum of 20 different types of products and a minimum of 10 different customers
- you can use the existing datafill provided in the base code

Insufficient datafill means that your assignment cannot be adequately tested; this may incur deductions of up to 100% of the assignment

Constraints

- your program must follow the existing design of the base code, including the encapsulation of control, UI, entity and array object functionality
- do not use any classes or containers from the C++ standard template library (STL)
- do **not** use any global variables or any global functions other than **`main`**
- do **not** use structs; use classes instead
- objects must always be passed by reference, not by value
- your classes must be thoroughly documented in every class definition
- all basic error checking must be performed

Submission

You will submit in *cuLearn*, before the due date and time, the following:

- a UML class diagram (as a PDF file) that corresponds to the program design
- one **`tar`** file that includes:
  - all source and header files for your program
  - a Makefile
  - a readme file that includes:
    - a preamble (program and modifications authors, purpose, list of source/header/data files)
    - compilation, launching and operating instructions

If you are working with a partner:

- only **one partner** submits the assignment, and the other partner submits **nothing**
- the submitting partner must enter the names of both partners in the **Online Text** box of the *cuLearn* submission link
- the readme file must contain the names of both partners

Grading

Marking components:

- UML diagram: 25%
  8 marks: Control object
  - 2 marks: correct relationship with UI object
  - 2 marks: correct relationship with entity object(s)
  - 4 marks: correct operations

  6 marks: Correct relationship between entity objects
  5 marks: Correct attributes and operations on the UI object
  6 marks: Correct attributes and operations on the entity objects
- Purchase class: 5%
  5 marks: correct attributes and operations on Purchase class
- Purchase collection class: 25%
  5 marks: correct attributes on purchase collection class
  20 marks: correct implementation of operations on purchase collection class
- Product purchase feature: 35%
  5 marks: correctly checks for existing customer
  5 marks: correctly prompts for multiple product ids
  5 marks: correctly checks for existing product and that stock is available
  5 marks: correctly keeps tally of amount and loyalty points
  5 marks: correctly reduces stock available and modifies customer's loyalty points
  5 marks: correctly *invokes* function for adding/incrementing data each product in purchase collection
  5 marks: correctly prints out total amount of purchases and points earned
- Print customers feature: 10%
  10 marks: correctly prints all purchase data for each customer to the screen

Execution requirements:

- all marking components must be **called**, and they must **execute** successfully to receive marks
- all **data** handled by marking components must be printed to the screen for marking components to receive marks

Deductions:

- Packaging errors:
  - 10 marks for missing Makefile
  - 5 marks for missing readme
  - 10 marks for **consistent** failure to correctly separate code into source and header files
  - 10 marks for bad style or missing documentation
- Major programming and design errors:
  - 50% of a marking component that uses global variables, global functions, or structs
  - 50% of a marking component that consistently fails to use correct design principles
  - 50% of a marking component that uses prohibited library classes or functions
  - 50% of a marking component where unauthorized changes have been made to provided code
- Execution errors:
  - 100% of a marking component that cannot be tested because the code does not compile or execute
  - 100% of a marking component that cannot be tested because the feature is not used in the code
  - 100% of a marking component that cannot be tested because data cannot be printed to the screen