cs3307 – Object oriented analysis and design

C++ Assignment: A Banking System

(Individual assignment)

Introduction:

This is an individual assignment. You may discuss concepts with colleagues in general terms **but the actual program design, code, and documentation must be your own**.

This assignment is to build an **Object-oriented** program through the implementation of a banking system in C++.

The banking system enables customers of a financial institution to perform financial transactions (such as: cash withdrawal, deposit, transfer funds from one account to another, etc.) without the need for a human cashier. In this respect, it is a like an automated teller system (ATM). However, the banking system is more than an ATM as there are other agents involved, such as the bank manager and a system maintainer, as described below.

The following is a description of the approximate system. You are to make it more precise by documenting the specific requirements the system will implement. You can add desired features that fall within the spirit of a bank. You can interpret the given requirements so as to give the system a specific behaviour. This flexibility permits you to experiment with the C++ language and get some experience with it.

System description:

The bank has an unspecified number of *customers*. For demonstration purposes, you can have a manageable number of test customers. The term "client" is a synonym for the term "customer". Each customer has either or both of a savings account and a chequing account.

There is a bank *manager* who has managerial powers to *open* and *close* an account and see the critical details of a particular, or all (at once), customers in a formatted display.

When a client wants to open a new account at the bank, it is assumed that s/he will contact the manager (e.g., in a face-to-face meeting) and ask him/her to create the desired account. Therefore, clients do not have the power to open a new account.

Once a customer has been given an account by the manager, s/he has access to use *only* his/her account. In a given user session with the bank, the customer will be provided the capabilities to use multiple operations in that session (e.g., withdraw funds from the chequing account, transfer funds to the deposit account, print recent transactions, etc.).

Each user of the system, regardless of his/her role type, has an "id" for secure login. After logging into the system, s/he will be bounded by his/her role as to what s/he can do with this system.

The banking system has the following core features (remember this is approximate!):

- Open/Close an account; close restricted to zero balance accounts.
- Deposit to, and withdraw money from, an account.

- Give warning messages for not sufficient funds (e.g., when withdrawing).
- Transfer a sum from one account to another.
- Obtain account balances.
- Give a warning message to the client if the balance on his/her “chequing” account will drop below the threshold of $1,000 – prior to him/her executing the operation. If s/he decides to go ahead despite the warning message, a charge of $2.00 is levied on the client for each such transaction. This levy is applied only once, when there is a crossover from above to below the threshold.
- Upon a client’s assumed face-to-face meeting with the manager to open an account, the manager will open either a savings or a chequing account, or both. Later, the client can request the manager to add the complementary account at anytime.
- The bank manager can display the account details of any given customer, or all customers, and obtain aggregate data on the funds in the bank, etc., categorised appropriately.
- Funds in the account transcend the duration of a particular user session with the bank.

The interaction with the system is through the text-based menus, so as to simplify the user interface.

There is also a role of a systems *maintenance person* who also has an “id” to access the system. S/he can turn ON/OFF the system’s *execution trace*.

An execution trace is a log of the system’s functions entered, along with the time entered, during the use of the system by the user. In order to associate a particular trace with the user that generated it, the user is identified at the head of the execution trace.

When the trace switch is set to ON, an execution trace is written, for a given account, in an external file that can be printed out at a later time by the maintenance person. No traces are generated when the switch is in the OFF state.

IMPORTANT: This assignment is not the time to get overly fancy with interface design, there is absolutely no need for overdone interfaces, such as implementing a Qt GUI. Please do not spend extraordinary time making an interface when core requirements are not met. Assignments will be marked based upon their meeting the requirements, whether or not it has a fancy interface (beyond the defined user-system interactions)

IMPORTANT:
Minimum set of requirements: The above set of requirements are a minimum set of requirements for which, if properly implemented, will obtain 70% of this assignment marks.

Enhancements: For a further 30% (to make up 100% of this assignment), you are to enhance the minimum set of requirements by another four orthogonal (i.e., distinct or new) requirements of your choice. These requirements should not be minor tweaking of the requirements in the minimum set – as interpreted by the instructing team whose decision is final and binding.
Please note that marks will only be given for fully functioning of software corresponding to the enhancement requirements in question.

Project requirements:

[Note: *The maximum possible marks for each deliverable are indicated in the square brackets below.]*

1) **Deliverable 1 – minimum set of requirements**: [8 max] Document textually the minimum set of requirements, categorised by user roles (customer, manager, maintainer, etc.), that have been implemented and are demonstrable by the execution of the system. Each requirement should be uniquely identifiable with an "id".
2) **Deliverable 2 – enhancement requirements**: [4 max] Identify enhancement requirements in written form in a similar manner.
3) **Deliverable 3 – System code and executable**: [50 max: 35 minimum set; 15 enhanced set]
   - Deliver C++ source code, along with any instructions on how to compile and run the program and any system configuration parameters.
   - Make the system platform explicit (Windows, Mac, Linux, etc.).
   - Deliver the executable file.
4) **Deliverable 4 -- Scenarios**: [30 max – 6 marks per scenario fully functional]
   - Document **five** scenarios in **a bullet point** form. Example:
     - Client logs into the ATM.
     - Selects his/her Deposit account.
     - Checks the balance.
     - Deposits CAD 20.
     - System deposits the funds.
     - System displays the resultant balance.
   - Each scenario must have an appropriate title (e.g., **Depositing funds**).
   - For each scenario, show **using screenshots**, the behaviour of the system as observable from the outside.
   - There should be at least one scenario per role (client, manager, maintenance person).
5) **Deliverable 5 – Implementation of the enhanced requirements**: [5 max]
   - Describe how the enhanced requirements (see Deliverable 2) have been implemented. In this description, please include names of the relevant software elements (class names, method names, data structures, etc.) so as to give a convincing answer. The length of this description is approx. half a page.
6) **Deliverable 6**: [3 max] Please describe, as bullet points, what you have learnt about developing the program in C++.
7) All documents should be readable on a Windows machine with standard application programs.
8) **Packaging, and Submission format:**
   - One PDF document (file) describing Deliverables 1,2,4,5 and 6.
     - Cover page for the whole document includes:
       - Course number and name;
       - "C++ Assignment: The Banking System"
       - Student's first and last names, last four digits of the student ID, email address.
       - Month and year
     - Each deliverable must start on a new page.
     - Header should include stuednt's first and last names.
     - Footnote should include page number **of** max-number-of-pages.
     - Name of the PDF file:
       <Student's last name and first name>_C++_Deliverables 1,2,4,5,6
   - One folder containing Deliverable 3.
     - Folder name: <Student's last name and first name>_C++_Deliverable 3

- In a zipped folder, include:
    - The PDF document
    - The folder
    - Zipped folder name:
        - <Student's last name and first name>_C++_Assignment

9) **Deadline:** 19th October – as specified on OWL.
10) Submitted in the OWL system.

Have fun!