CMPS 260

Spring 2005

Project #2

2005.01.30

Date Assigned: **Friday, February 4, 2005**
Due Date: **10:00 PM, Thursday, February 17, 2005**

The coded solution to the following problem is to be done by you and only you. You may ask help from the class teaching assistants and the instructors, but you may not ask for help on this project from anyone else. You may use your notes, C++ texts, on-line tutorials, etc., but the code must be your own.

1. **Project Description:**

   Simple Bank Company runs a “piggy bank” service. Customers can put money in their accounts and take money out of their accounts. The money in an account receives no interest and an account balance cannot fall below 0. Customers access their accounts by logging into and using a menu on an automated teller machine. New accounts can be created at any time via the automated teller and an initial deposit.

   As the chief and only programmer for SBC, you are to create an object oriented solution to service customers' requests and maintain accounts.

2. **Design Requirements:**

   The solution will use objects of two classes that you will design and implement. One class will model a user account. The other class will model the automated teller.

   The program solution will use an array of account class objects to represent the customer accounts. The program must keep track of both the array elements in use and the number of array elements that are actually in the array. Users are not permitted to create accounts if there is no room in the array.

   The program solution will use one object of the automated teller class to communicate with customers. The teller class will have no actual knowledge of the user's account. (I. e. the automated teller is a “front” or “interface” for the bank. It is not the bank.)

   The program solution will load all account information from a file when the program begins and save all account information back to that file when the program ends. The program will halt when the user “9999” enters the password “9999” into the automated teller login screen.

Automated Teller Class

The automated teller class will need to have public methods (public member functions) to:

- display a greeting menu that allows a customer to select to log in or create a new account
- log in by entering his or her account name and password
- create a new account by entering a new password, account name and initial deposit
- display whether a requested account was created or not
- display a menu that allows a user to select to see the account balance, deposit to the account, withdraw money from the account and to log out
- display the account balance
- accept a deposit
- accept a withdrawal request
- reply to a withdrawal request indicating success or failure

Customer Account Class

The customer account class will need to have private data members to maintain:

- account name
- account password
- account balance

The customer account class will need to have public methods (public member functions) to:

- set the account name
- set the password
- set the account balance, which must be never less than 0
- get (i. e. return) the account name
- get (i. e. return) the account password
- get (i. e. return) the account balance
- deposit a passed amount to the account balance
- withdraw a passed amount from the account balance, but not allow a resulting balance of less than 0; this method must return true if the withdrawal was permitted, false if the withdrawal was not permitted

The customer account class will need to have a default constructor to be called when the array is created in order that all private data members are initialized to zero or blank. Optionally, the class may also have a constructor to set the values of the account name, password and balance from parameters. In this case, the three set methods may be omitted.

3. Additional Requirements:

- Each account name and password combination must be unique

- You are responsible for following the requirements as given in documents reachable from the class web site via the **Minimum Documentation** and **Naming Conventions** links.

- Your program may have global constants but may not have global variables.

- Your program solution must use good programing style. For example, call a function load the value in the data file into the array of account objects at the start of the program, then call a function to at the end of the program to write the values in the array elements back to the file. In between, call another function to handle the logic necessary to maintain communicate with the user and maintain the accounts. To rephrase, try to make your main function an outline of the logic of the program solution, one that uses calls to functions and object methods to direct the order of the operations rather than trying to write all operations in the main.

- Methods in your class definitions must has accompanying pre-condition and post-condition documentation. (See the link **Additional Documentation** on the class web site.)

- Each class design is to be placed in a file named for the class and ending in ".h" (a header file). Each class implementation is to be placed in a file named the same as the file of the design, but ending in ".cpp". Each header file is to include wrappers to prevent unnecessary recompilation

- A makefile is to be created for and submitted with this assignment.

4. Submitting: You are responsible for submitting your work both electronically and via a hard copy.

- Instructions for submitting an electronic copy of your project can be found on the class web site, via the **Submitting Your Work** link. When prompted, name your project "proj2".

- A hard copy of the code in your project is due the school day after you submit electronically. Place the print out of your code in a manila folder, write your name, clid and section on the folder, then bring the folder to class or your instructor's or TA's office.