**File Name: Project 04 Design Document**

**Authors: Juan Chavarria**

**Date: 12/01/2023**

**Assignment Number 5**

**CS 105C Fall 2023**

**Instructor: Dr. Palacios**

**Classes**

**Account**

This class serves as a base function for the two different types of accounts in our project: savings and checking. This class is inherited by those two other classes, and it has the basic functionalities of all bank accounts: depositing, withdrawing, calculating interest, and having an end of month summary. The other classes will use this function with added steps, but it is important to note that this is the foundation of the system. The class members are below:

Member Structure: Most functions and variables are left public for easy access from the Longhorn.cpp file.

Public:

* Float balance: account balance throughout the month
* Float deposits: total number of successful deposits made
* Float withdrawals: total number of successful withdrawals made
* Float dTotal, wTotal, iTotal, chTotal: total monetary value accumulated through deposits, withdrawals, interest, and monthly charges.
* Float rate: annual interest rate of the account
* Float charges: total monthly charges to an account
* Account(float b, float r): constructor that sets b, the account balance, and r, the annual rate to the “balance” and “rate” variables
* virtual void deposit(float): virtual base function that takes in a float and increases the balance by that amount. It also increases “deposits” variable by 1, and it adds the deposit amount to dTotal.
* virtual void withdraw(float): virtual base function that takes in a float and decreases the balance by that amount. It also increases “withdrawals” variable by 1, and it adds the withdrawal amount to wTotal.
* void monthlyProc(): function that subtracts the monthly charges from the balance and calls calcInt() to calculate the monthly interest. It also resets the withdrawals, deposits, and charges values.

Protected:

* void calcInt(): function that calculates and adds the accumulated monthly interest of accounts with a positive balance.

**Savings**

This class serves as a representative of a savings account. It has a public relation to the Account class, and it inherits the member variables shown in the previous sections. The addition to the class is that it now considers the status of the savings account, which changes whenever its balance goes below/above 25. This status affects withdrawals, since we now cannot withdraw if the account is inactive. This account also has a special monthly $1 charge for more than 4 total withdrawals. Having an account status also means that we must calculate it after all deposits, withdrawals, and monthly charges. The class members are below:

Member Structure:

Public:

* bool status: variable that indicates if a class is active or inactive based on a balance threshold of $25
* Savings(float b, float r): constructor that initializes the balance member of the Account class to b, and the rate member to r. The status of the account is checked upon call (if greater than or equal to 25, the account is activated. If less than 25, it’s inactive)
* void withdraw(float): this method overrides the withdraw method of Account. The withdrawal goes through only if the account is active. If it is, then the withdrawal is made by calling the base class withdraw() function with the float passed by value, and the status of the account is set after the transaction.
* void deposit(float): this method overrides the deposit method of Account. The only difference is that before depositing, the status of the account is checked. This is then used to change the status if the deposit brings it above a balance of $25.
* void monthlyProc(): this method overrides the base class method by adding the proper service charge fees (if the user makes more than 4 withdrawals, a $1 fee is added for each one above 4). Then the base method is called. After it’s called, the status is checked to see if the account is still active

**Checking**

This class serves as representative of a checking account. It has a public relationship to the Account class. The checking account doesn’t have a status variable. The only differences from the base class are in the withdraw() method and the monthlyProc() service charges. The class members are below:

Public:

* Checking(float b, floatr): constructor that initializes the balance member of the Account class to b, and the rate member to r.
* void withdraw(float): this method overrides the withdraw method of the base class and adds the extra step of charging a $15 service fee if the user tries to withdraw more money than they have in their account. Withdrawals larger than the balance aren’t allowed as well.
* Void monthlyProc(): this method also overrides the base class method. It first calculates the monthly fees by adding the monthly $5 fee plus a $0.10 fee per withdrawal. After that, it calls the base monthlyProc()

**Main File Implementation**

The approach I took was the following:

1. Process the account type and the initial balance & rates using the createAcc() function, which returns a pointer to an account object with the appropriate savings/checking account.
2. Once the account is made, the transactions of the account are read in a while loop and processed iteratively using displayTran(), which calls either a withdrawal or a deposit and writes to the output file the transaction statistic.
3. The monthStats() function is called to print the end of month stats of the account.
4. Steps 1-3 are repeated for the second account

In order to simplify the implementation of the monthStats() function, I added 4 member variables to the account class: dTotal, wTotal, iTotal, chTotal. These variables allowed me to keep track of all the successful transactions within the class implementation, which reduced the amount of code needed in main. Also, making the member structures public allowed easy access of account variables, such as the balance and the other transaction statistics.

**Main() Functions**

createAcc(float& bBal, string& accType): processes the first two lines of the input text file. This includes the type of account, the beginning balance, and the interest rate of the account. This then writes to the output file, and dynamically allocates memory for a checking or account class and assigns it to an Account pointer, which is returned and used in main.

displayTran(Account\*, float&, string&): processes a transaction float, and calls a withdrawal or a deposit accordingly. It also prints to the result file if the transaction was invalid or successful.

monthStats(Account\*, float&): this function calls the monthlyProc() method of account, which calculated the end of month statistics. Then, it writes all the results to the output file of the monthly statistic, such as the beginning and final balance, the deposit totals, withdrawal totals, interest and service charge fees.

**Assumptions**

I assumed that the file structure was going to be:

\*type of account (checking, savings) MUST BE LOWERCASE\*

Balance (float) interest rate (float)

Transaction

Transaction

\*type of account (checking, savings)\*

Balance (float) interest rate (float)

Transaction

Transaction

One transaction is typed in one line respectively, and it uses a “-“ when a withdrawal is made, and no sign when a deposit is made. You can write any number of transactions, but once they are done, you must separate the next account with one empty line and continue with the next account using the same format. The last assumption about the input file is that there will only be two accounts input.

Another assumption is that the user will never be able to withdraw more than their balance in a savings account (a user can never have a withdrawal that will make their account be negative).

Another assumption is that the user will always be able to deposit into either type of accounts (even to a inactive savings account).

The last assumption is that there will never be a transaction where the user neither deposits nor withdraws $0.00.

These assumptions helped me structure the text file and make the data processing fairly easy.

**Instructions for Compiling:**

Get all the code submitted (Longhorn.cpp, Account.h, Account.cpp, Savings.h, Savings.cpp, Checking.h, Checking.cpp, transactions.txt, makefile) in one folder. In your terminal, go to the folder where everything is saved (ex: cd “Project 4”), and then type “make” in your terminal. This will create a link all the files and create a.out, which is the executable file. Then, run “./a.out” in the terminal. This should execute the linked file and run it.