CS 1410 Introduction to Computer Science - CS2
Assignment #6
Given:
Due:
Total Points: 50 points

Problem description

The purpose of this assignment is to build an object-oriented program to process a list of bank transactions. Transactions include creating accounts, making deposits and withdrawals, adding accumulated interest, and deducting accumulated fees.

The following rules apply:

- A bounced check costs $25 and is deducted immediately from the checking account balance.

- There is a fee for each check (checking account withdrawal). Each checking account may use a different fee. The fees are charged as part of an update (see below).

- Savings accounts accrue interest. Each savings account may have a different interest rate. Interest is credited as part of an update (see below).

- An attempt made to withdraw more money than is available in a savings or checking account shall be rejected and a message indicating such shall be output.

The transaction file uses the following symbols: s, c, t, u, r, and d.

- **s** - Create a new savings account. An account number, a name, an initial balance, and the interest rate are supplied.

- **c** - Create a new checking account. An account number, a name, an initial balance, and the fee for each processed check are supplied.

- **t** - Perform a transaction. An account number and the amount of the transaction are supplied.
    - For savings accounts, a positive value is a deposit and a negative value is a withdrawal.
    - For checking accounts, a positive value is a deposit and a negative value represents a withdrawal via a check.

- **u** - perform an update. An account number is supplied.
    - For savings accounts, calculate and add interest earned.
    - For checking accounts, calculate and deduct the fee for each check.
    - Updates will only happen on an active account

- **r** - perform a transfer. Two account numbers are supplied
    - This will transfer the balance from one account to another
    - This will **close** the first account and create a new active account

- **d** - list the number of accounts. No supplied values
    - This option will list the total number of accounts
    - It will also list the number of accounts of each type

Use the list of transactions contained in bank.txt as input; there is no user input. **All transactions shall be processed in a single loop.** Assume that you wrote the program that generated the transaction file. Consequently, you know that all data are present and correct and that the formatting is consistent. This means that the file will always be formatted correctly. IE Accounts will always exist before an action is taken on them. You do not need to check input for errors.

Create a base class named Account. Created derived classes named Savings and Checking.

Overload the stream insertion operator (<<) and use it for output. Do not output from any class.

Task 1 (14 points)

- Create the Account base class
    - The base class must contain all shared data variables
        - A protected type value must be stored that distinguishes the Account type
        - A protected 'active' state value
        - The name data type must be a char*
        - Add any other needed variables
    - A constructor must be created that allows the setting of the account type variable
    - An overloaded equality operator function. The function should use the account number, account type, and active state for equality checking.
    - Create accessors for the active state
    - Implement a static function 'getNumberOfAccounts()'. This function should return the total number of accounts.

Task 2 (11 points)

- Create the Savings class
    - Implement the constructor
        - It must pass the type using the base class constructor
    - Implement the 'handleTransaction' function
    - Implement the 'handleUpdate' function
    - Implement the operator << function

    - A copy constructor must be created for transfers
        - This will copy the current balance and all other data members
    - Implement the equality operator. This function should use the base class equality and also check the new data members.
    - Implement a static function ‘getNumberOfSavingsAccounts()’. This function should return the total number of savings accounts.

Task 3 (10 points)

- Create the Checking class
    - Implement the constructor
        - It must pass the type using the base class constructor
    - Implement the ‘handleTransaction’ function
    - Implement the ‘handleUpdate’ function
    - Implement the operator << function
    - A copy constructor must be created for transfers
        - This will copy the current balance and all other data members
    - Implement the equality operator. This function should use the base class equality and also check the new data members.
    - Implement a static function ‘getNumberOfCheckingAccounts()’. This function should return the total number of checking accounts.

Task 5 (15 points)

- Create a Bank class to handle all actions
    - The class should contain the following
        - A private list(s) of accounts
        - Functions for creating each account type
        - Private function(s) for finding an account
        - Functions for handling all supplied actions. For example transactions, updates, creates, lists, etc
- Load the symbols from the file
    - The file name must be loaded from the command line

Extra Credit (5 points)

- Build a user interface to the Bank software that will allow a Teller to interact with the system. The interface should validate inputs and allow all commands to be executed. This should not break the file loading functionality!

Example Input

s 0 Dessie 1000.00 0.05
c 1 Andrew 400.00 0.25

```
t 1 -200.00
t 1 642.00
t 0 2000.00
t 1 -300.00
u 0
t 1 -100.00
t 1 -400.00
u 0
t 0 3000.00
u 1
t 1 -500.00
u 0
u 1
r 1 4
d
```

Example Output

```
new account...
Dessie (savings): balance is $1000.00

new account...
Andrew (checking): balance is $400.00

transaction...
Andrew (checking): $-200.00 applied to account
Andrew (checking): balance is $200.00

transaction...
Andrew (checking): $642.00 applied to account
Andrew (checking): balance is $842.00

transaction...
Dessie (savings): $2000.00 applied to account
Dessie (savings): balance is $3000.00

transaction...
Andrew (checking): $-300.00 applied to account
Andrew (checking): balance is $542.00

update...
Dessie (savings): $150.00 applied to account
Dessie (savings): balance is $3150.00
```

transaction...
Andrew (checking): $-100.00 applied to account
Andrew (checking): balance is $442.00

transaction...
Andrew (checking): $-400.00 applied to account
Andrew (checking): balance is $42.00

update...
Dessie (savings): $157.50 applied to account
Dessie (savings): balance is $3307.50

transaction...
Dessie (savings): $3000.00 applied to account
Dessie (savings): balance is $6307.50

update...
Andrew (checking): $-1.00 applied to account
Andrew (checking): balance is $41.00

transaction...
Andrew (checking): $-500.00 rejected (insufficient funds)
Andrew (checking): $-25.00 applied to account
Andrew (checking): balance is $16.00

update...
Dessie (savings): $315.38 applied to account
Dessie (savings): balance is $6622.88

update...
Andrew (checking): $-0.25 applied to account
Andrew (checking): balance is $15.75

transferring...
Andrew (checking): $15.75 transferred to checking account 4
Andrew (checking): balance is $15.75

listing all accounts…
There are a total of 3 accounts.
There are 1 savings accounts
There are 2 checking accounts