Assignment #7 – All Together Now

Description:

Write a program to create bank accounts and manage banking transactions. This program will demonstrate the principles of inheritance, exceptions, and application of the standard template library. It will read from two input files and output the transaction results to an output file. The first file has a list of accounts, each with initial data for the opening date (month day year), account number, type of account, opening balance, and account owners name. The second file contains the transactions the program will need to process, each with a date (month day year), account number, amount, transaction type, and optionally a second account number if the transaction is a transfer. Your program should read three file names separated by whitespace representing the accounts file, the transaction file, and the output file. Your source code must be fully documented per the course rubric for assignments (see Canvas homepage).

Provided code:

You are provided with the *Account* class, the *Date* class, a header *mainheader.h* for inclusion in your main (you should not include any other #includes other than for debug), and a couple functions in *functions.cpp* to aid in printing the transactions output from your main to the output file (note: the provided source code is not documented per Canvas but your code must be documented per Canvas)

The *Account* class is the base class for all accounts, which has the following attributes: account owner's name, account number, account opening date and account balance; and methods for getting, setting and display the name, number, date and balance; and transactions depositing, withdrawing, and transferring funds. If an error occurs with any transaction (see example output below), then the appropriate method throws a *TransactionError* exception (see below).

Classes to be developed:

There are four types of *Accounts*: *Generic, Checking, Savings*, and *MoneyMarket*. The *Generic* account is simply an *Account* (i.e. it is an instance of the base class Account). The *Checking* and *Savings* accounts derive from *Account* (i.e. they are derived classes from the base class *Account*). The *Savings* account has a data member *interestRate* which is currently **10%**. The *Savings* account has a derived class *MoneyMarket* that has an *interestRate* which is currently **20%**.

1) Create a class called *TransactionError* to handle exceptions from the *Account* related and *Bank* classes.
   a) This class must be created before any other class since it is used by *Account*.
   b) This class should have a string private data member, a constructor that takes a string (the error message) as a parameter, and a *what()* method that returns the string.
   c) Use inline method definitions so that only the header file *TransactionError.h* is required.
2) Create a class called *Savings* that is derived from the *Account* class.
   a) *Savings* accounts will have no penalty/cost for any transactions

   b) Interest will be added at the first of each month based on the balance amount at the end of the previous month. This should be updated with each transfer, deposit, or withdrawal. (hint: override *UpdateAcct()* to handle interest)
   c) The base class constructor (i.e. *Account*) must be called as part of this class's constructor implementation heading (see slides)

3) Create a class called *MoneyMarket* that is derived from the *Savings* class.
   a) *MoneyMarket* accounts will have a withdrawal cost of **$1.50** for each withdrawal (hint: override *Withdrawal()* to handle fees).
   b) The base class constructor (i.e. *Savings*) must be called as part of this class's constructor implementation heading (see slides)
4) Create a class called *Checking* that is derived from the *Account* class.
   a) *Checking* accounts will have no withdrawal cost and can be overdrawn by up to **$200**.
   b) However, each withdrawal that results in a negative balance will result in a **$20** fee.
      i) If the account would go beyond the $200 overdraw limit then the transaction should not be completed and a *TransactionError* exception should be thrown with an appropriate error message (see example output below). (hint: override *Withdrawal()* to handle overdraft and associated fees).
   c) The base class constructor (i.e. *Account*) must be called as part of this class's constructor implementation heading (see slides)
5) Create a class called *Bank* to create and manage a list of accounts using the STL *list*
   a) Recall that polymorphism requires the *list* to be a list of pointers to *Account* (the base class) objects
   b) Must have methods to open an account (*OpenAccount*) and for *Deposit*, *Withdraw*, and *Transfer* transactions. All methods except opening (i.e. creating) an account must use pointers to objects of the base class *Account* (i.e. they must support dynamic polymorphism)
      i) Opening an account must create dynamic objects (i.e. pointers to) for each account type and add them to the list
   c) All constants related to interest rates, fees, and withdrawal limits must be declared in *Banks.h* and only used in *Banks.cpp* (hint: constructors and/or methods for the relevant accounts must accept their relevant constants passed to them by Bank's open account method)
   d) At no time should the list be exposed (i.e. it is *private*)
   e) There should be a *private* helper method called *Find* to find an account in the *list*
      i) If the account is not found, then *Find* should throw a *TransactionError* exception (see below for the appropriate message).

Main:

The main program processes a number of account transactions from input files and outputs the results to an output file (your main will read the file names and open them as appropriate as described above). <u>Main can only use the *Bank*, *Date*, and *TransactionError* classes</u> to process information. You must use the *Bank* class for account creation and transactions (i.e. main does not have access to any of the *Account* classes). For each transaction you will be required to output the account information, transactions type, date and balance. All errors will be thrown by classes outside of main but main <u>must</u> catch the *TransactionError* exception. The action to be taken with a caught exception is to output an error message (that includes using the *TransactionError* *what()* method with some preceding text – see the output example below) and then to continue reading transactions from the input file (i.e. continue the transaction processing loop).

Example input files and output file (note that there are no column headings for the input files)

Accounts input file:

```
Open Date     Acct#   Acct Type     Amount    Name
4   1  2021   6789    Generic       100.00    Joe Big
4   1  2021   2323    Checking      50.00     Jennifer Kim
4   1  2021   1212    Savings       300.00    Nery Chapeton Lamas
4   1  2021   3434    MoneyMarket   100.00    Shannon Alfaro
```

Transactions input file:

```
Open Date     Acct#   Amt.          Type         TxAcct#
5   1  2021   1212    100.00        Deposit
5   1  2021   2323    100.00        Deposit
5   1  2021   3434    100.00        Deposit
5   1  2021   6789    50.00         Deposit
5   1  2021   1234    50.00         Deposit
6   1  2021   1212    200.00        Withdrawal
6   1  2021   2323    200.00        Withdrawal
6   1  2021   3434    50.00         Withdrawal
6   1  2021   6789    10.00         Withdrawal
6   1  2021   4321    50.00         Withdrawal
7   1  2021   1212    250.00        Transfer     2323
7   1  2021   2323    80.00         Transfer     3434
7   1  2021   3434    300.00        Transfer     1212
8   1  2021   1212    50.00         Transfer     2323
9   1  2021   1212    100.00        Deposit
9   1  2021   2323    100.00        Deposit
9   1  2021   3434    100.00        Deposit
10  1  2021   1212    300.00        Transfer     1212
```

Output file:

```
TRANSACTION         DATE       ACCT #  ACCT NAME                 AMOUNT        BALANCE    FROM ACCT#    FROM ACCT BAL
------------        ---------- -----   --------------------  ----------  -------------    ----------    -------------
OPEN GENERIC         4/1/2021   6789    Joe Big              $   100.00  $       100.00
OPEN CHECKING        4/1/2021   2323    Jennifer Kim         $    50.00  $        50.00
OPEN SAVINGS         4/1/2021   1212    Nery Chapeton Lamas  $   300.00  $       300.00
OPEN MONEY MARKET    4/1/2021   3434    Shannon Alfaro       $   100.00  $       100.00

 Deposit             5/1/2021   1212    Nery Chapeton Lamas  $   100.00  $       430.00
 Deposit             5/1/2021   2323    Jennifer Kim         $   100.00  $       150.00
 Deposit             5/1/2021   3434    Shannon Alfaro       $   100.00  $       220.00
 Deposit             5/1/2021   6789    Joe Big              $    50.00  $       150.00

*** ERROR FOR TRANSACTION: Deposit, Account 1234 not found ***

 Withdrawal          6/1/2021   1212    Nery Chapeton Lamas  $   200.00  $       273.00
 Withdrawal          6/1/2021   2323    Jennifer Kim         $   200.00  $       -70.00
 Withdrawal          6/1/2021   3434    Shannon Alfaro       $    50.00  $       212.50
 Withdrawal          6/1/2021   6789    Joe Big              $    10.00  $       140.00

*** ERROR FOR TRANSACTION: Withdrawal, Account 4321 not found ***


*** ERROR FOR TRANSACTION: Transfer, Insufficient funds in account 2323 ***

 Transfer            7/1/2021   2323    Jennifer Kim         $    80.00  $        10.00    3434          $       173.50
 Transfer            7/1/2021   3434    Shannon Alfaro       $   300.00  $       473.50    1212          $         0.30
 Transfer            8/1/2021   1212    Nery Chapeton Lamas  $    50.00  $        50.33    2323          $       -60.00
 Deposit             9/1/2021   1212    Nery Chapeton Lamas  $   100.00  $       155.36
 Deposit             9/1/2021   2323    Jennifer Kim         $   100.00  $        40.00
 Deposit             9/1/2021   3434    Shannon Alfaro       $   100.00  $       781.84

*** ERROR FOR TRANSACTION: Transfer, Can not transfer from/to the same account 1212 ***
```