150 pts

Name: ___________________________________

Class Day / Time: ___________________________________

Due Date: ___________________________________

Assignment #6 – Saddleback Bank - OOP

This assignment will use object oriented programming to represent banking accounts and manage banking activities. You have been hired by Saddleback Bank to develop a program to manage its bank accounts.

In this assignment you will be developing a set of classes to simulate various types of bank accounts and demonstrate the principles of inheritance.

First, 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; depositing, withdrawing, and transferring funds.

The Account class has two derived classes, Checking and Savings accounts. The Savings account has an attribute, interestRate, which is currently 10%. The Savings account has a derived class MoneyMarket that also has an interestRate attribute, which is currently 20%.

The main program will use the class defined above and process a number of account transactions from an input file. Each transaction will include the account number, transaction date and amount. You will output all transactions and account balances in an output file (see sample output file for further details). Have a generic function, which will implement deposit, withdrawal, transfer, or print.

Implement this using a linked list of objects – create a Bank class to manage those objects.

Develop a set of classes to simulate various types of bank accounts.

a. You will create a class to handle the dates including day, month and year. (The Account class will have an is-composed-of relationship with this class so develop it first).
b. Create an Account class, which is the base class for all accounts that has public methods to get and set the name, account number, date, and balance.
   →To make it easier to use - **overload** the set methods`. There should also be methods to withdraw, deposit, and transfer funds from one account to another.
c. Create classes for Checking and Savings that are derived from the Accounts class.
d. Create a class for Money Market is derived from the Savings class.

INCLUDE THE FOLLOWING PUBLIC METHODS
(you may use additional methods if necessary)

- **Constructor**
  In the Account class, the default constructor sets the value of balance to 0 dollars and 0 cents. You will need to have a method to initialize the other attributes of the class.

- **Deposit**
  The deposit method simply adds the deposit amount to the current balance.

- **Withdraw**
  The withdraw method is designed to simply subtract the withdraw amount for the current balance, but it should be a virtual method so that the modified versions of the withdraw method will be called for each of the derived classes. The withdraw method also checks to make sure there is enough money in the Savings and Money Market accounts to make the withdrawal. If there are insufficient funds, an error message is displayed. This method should return a **bool** (true = a valid transaction, false = an invalid transaction. The three specific account types (not including the base Account class) will implement withdrawal in the following ways:

  - **Savings** accounts will have no penalty.
  - **MoneyMarket** account involve a cost, each withdrawal costs $1.50.
  - **Checking** Accounts will allow the user to overdraw up to - $200. However, each withdrawal in a negative balance will result in a $20 fee. If the account will go below $200 output an error message and return false.

- **Transfer**
  The transfer method specifies the account to transfer from in the first argument – by the amount specified in the second argument into the account of the object, which is calling the method. It checks to make sure that the destination and source accounts are not the same. Transfers can only be made if the from account has sufficient funds, otherwise output an error message. Also, you can't transfer from the same account you are transferring into, output an error message in this case as well. This method should return a bool to indicate if the transaction went through (true = a valid transaction, false = an invalid transaction. Transfers are treated like withdrawals in the "from" account and deposits in the "to" account.

- **Print**
  The print method simply displays the account number and the current balance in the correct form (with the $ sign and to only two decimal places). You will need to include a feature in each method that modifies the account balance to update the balance if there is interest in the account since the last time you accessed the account.

Additional requirements:

- Interest rate will be added to each account 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, withdrawal or print. You will need to pass in the Date function to handle this. **Hint:** To handle the interest based account you should include the last access date (as another attribute) to allow you to update the interest and have a record of the last date the account has been updated.

- Overdrawn checking accounts will be penalized $20 a month.
- Account numbers should be validated where appropriate

THE BANK CLASS

This class will create and manage a linked list of accounts. It will have methods to deposit, withdrawal or transfer. There should also be a method to Find an account. You may want to add to extra non-public methods to find an account and return a pointer and to output the list for testing. Accounts can be added to the front of the list. At no time should the list be exposed by returning a link to it. The Account class should hold the next variable, but it should not be public – create get and set methods to access it.

THE MAIN PROGRAM

Create a main program that will be using the classes above and processing information from two input files. The first file has a list of accounts and initial data for the accounts. The second file will contain the transactions the program will need to process. You will need to create dynamic objects for each account. Use polymorphism as necessary. For each transaction you will be required to output the account information, transactions type, date and balance (** **unless it was an invalid transaction** **). See sample output below.

Turn in (STAPLED IN THIS ORDER)

1 – The **FIRST PAGE** of this assignment as a coversheet
2 – Output as described below → cut and paste into an output.txt file within eclipse.
3 – header file for main followed by main.cpp
4 – Functions
5 – The header file followed by the implementation for each class – in this order.

- Bank Class first
- Account Class
- Checking
- Savings
- Money Market

ACCOUNTS INPUT FILE

```
Open Date  Acct#  Acct Type  Balance  Name
4 1 2012  2323   Checking    50.00   Jennifer Kim
4 1 2012  1212   Savings    300.00   Nery Chapeton Lamas
4 1 2012  3434   MM         100.00   Shannon Alfaro
```

TRANSACTIONS INPUT FILE

DATE Formatted
MM DD YYYY ACCT # AMOUNT TRANSACTION

```
5 1 2012     1212    100.00     Deposit
5 1 2012     2323    100.00     Deposit
5 1 2012     3434    100.00     Deposit
6 1 2012     1212    200.00     Withdrawal
6 1 2012     2323    200.00     Withdrawal
6 1 2012     3434    50.00      Withdrawal
7 1 2012     1212    50.00      Transfer
2323
7 1 2012     2323    80.00      Transfer
3434
7 1 2012     3434    300.00     Transfer
1212
9 1 2012     1212    100.00     Deposit
9 1 2012     2323    100.00     Deposit
9 1 2012     3434    100.00     Deposit
10 1 2012    1212    300.00     Transfer
1212
```

NOTE: FOR A TRANSFER

MM DD YYYY ACCT # AMOUNT Transfer
TRANSFER ACCT#
(Notice for a transfer the transfer account # is on the next line)

OUTPUT FILE

```
TRANSACTION        DATE          ACCT #   ACCT NAME                 AMOUNT         BALANCE       FROM ACCT#      FROM ACCT BAL
-------------      ----------    -----    --------------------    ----------    -------------    ----------      -------------
OPEN CHECKING      04/01/2012    2323      Jennifer Kim           $    50.00    $       50.00
OPEN SAVINGS       04/01/2012    1212      Nery Chapeton Lamas    $   300.00    $      300.00
OPEN Money Market  04/01/2012    3434      Shannon Alfaro         $   100.00    $      100.00

 Deposit           05/01/2012    1212      Nery Chapeton Lamas    $   100.00    $      430.00
 Deposit           05/01/2012    2323      Jennifer Kim           $   100.00    $      150.00
 Deposit           05/01/2012    3434      Shannon Alfaro         $   100.00    $      220.00
 Withdrawal        06/01/2012    1212      Nery Chapeton Lamas    $   200.00    $      273.00
 Withdrawal        06/01/2012    2323      Jennifer Kim           $   200.00    $      -70.00
 Withdrawal        06/01/2012    3434      Shannon Alfaro         $    50.00    $      212.50

*** TRANSFER FROM 2323 TO 1212 NOT ALLOWED! ***
***        DUE TO INSUFFICIENT FUNDS        ***

 Transfer          07/01/2012    2323      Jennifer Kim           $    80.00    $      -10.00         3434     $       173.50
 Transfer          07/01/2012    3434      Shannon Alfaro         $   300.00    $      473.50         1212     $         0.30
 Deposit           09/01/2012    1212      Nery Chapeton Lamas    $   100.00    $      100.36
 Deposit           09/01/2012    2323      Jennifer Kim           $   100.00    $       50.00
 Deposit           09/01/2012    3434      Shannon Alfaro         $   100.00    $      781.84

*** ERROR - CAN'T TRANSFER INTO THE SAME ACCOUNT ***
```