Assignment #6

Saddleback Bank - OOP

Introduction

- You have been hired by Saddleback Bank to develop a program to manage its bank accounts
  - Savings
  - Checking
  - Money Market
- In this assignment you will be developing a set of classes to simulate 3 types of transactions and demonstrate the principles of inheritance and polymorphism
  - Deposit
  - Withdrawal
  - Transfer

Input Files

- There are two types of files that the Bank Manager can process:
  - New Accounts
  - Transactions

- New Accounts will have the following format:

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
```

Input File Format

- Transactions File will have the following format

TRANSACTIONS INPUT FILE

```
DATE Formatted
MM DD YYYY   ACCT #   AMOUNT     TRANSACTION
TRANSFER TO ACCOUNT
```

```
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
```

Classes

- You will need to develop a set of classes to manage the bank, the transactions processed by the bank, and the various types of bank accounts
  - Date: This class will handle the date including day, month and year (Done!)
  - SaddlebackBank: This class will manage the bank and maintain the list of accounts
  - Account: This is the base class for all accounts that has public methods to get and set the name, account number, date, balance, and a list of transactions
    - To make it easier to use - overload the set methods
    - There should also be methods to
      - **withdraw,**
      - **deposit funds**
      - **Transfer**

Classes derived from Account

- Create classes for Checking and Savings that are derived from the Accounts class
  - Checking accounts
    - **earn no interest,**
    - **permitted to be overdrawn up to $-200.00**
      - » Incur $20 fees for each transaction resulting in a negative balance
    - **$20 fee each month the account is overdrawn**
  - Savings accounts
    - **earn 10% interest each month,**
    - **not permitted to have a negative balance**

Class Derived from Savings

Money Market Account is derived from the Savings class

- Money Market accounts
  - earn 20% interest each month
  - not permitted to have a negative balance.
  - Each withdrawal costs $1.50
- Interest earned for Money Market and Savings Accounts
  - 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 transaction

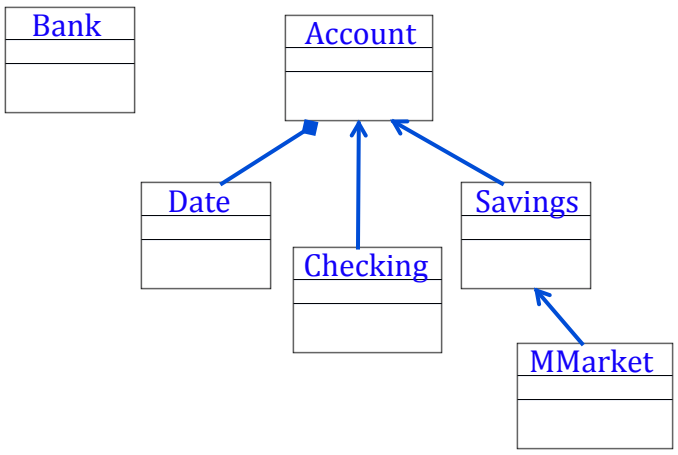
Transactions

- Deposit
  - Adds amount to the account provided
- Withdrawal
  - Removes amount from account
    - Must check balance
    - For checking accounts only
      - **Allow to overdraw up to $200**
      - **Charge fee if overdrawn**
    - Money Market Accounts
      - **Charge a fee for every withdrawal**
- Transfer
  - Withdrawals from the Account
  - Deposits to the To Account
- All accounts must be updated after the first of the month
  This can be done by checking the previous access date and verifying how many months have passed
  - Savings and Money Market Accounts → Add interest
  - Checking Accounts → may have fees

Exercise

- You will need to create the Class Definition for all the classes we discussed
  - Class SaddlebackBank
    - Account List
  - Class Account
    - Class CheckingAccount
    - Class SavingsAccount
      - **Class MoneyMarketAccount**
  - Class Date (already done!)

Class Design Diagram



Saddleback Bank Class

```
class Bank
{

public:
            /*****************************************************
                 CONSTRUCTORS & DESTRUCTOR
             *****************************************************/
            Bank();
            ~Bank();

            /*****************************************************
                 MUTATORS
             *****************************************************/
            void OpenAccount(Account *newAcct);
            bool Deposit(Date transDate, int acctNum, float amount);
            bool Withdrawal(Date transDate, int acctNum, float amount);
            bool Transfer(Date transDate, int acctNum, float amount, int transFromAcctNum);

            /*****************************************************
                ACCESSORS
             *****************************************************/
            // Output details on all accounts
            void OutputList() const;

protected:
            // Returns a pointer to an account
            Account *FindAccountPtr(int acctNum);

private:
        struct AccountList {
            Account     *customerAccount;
            AccountList *next;
        };

            AccountList *head;
            int         numberOfAccts;
};
```

Account Class

```
class Account
{

public:
            /*****************************************************
                 CONSTRUCTORS & DESTRUCTOR
             *****************************************************/
            Account();
            Account(Date openingDate, string name, int accountNum, float balance);
            virtual ~Account();

            /*****************************************************
                 MUTATORS
             *****************************************************/
            void SetAllValues(Date openingDate, string name, int accountNum, float balance);
            void SetValue(string name);
            void SetValue(int    accountNum);
            void SetValue(float  balance);
            void SetValue(Date   openingDate);

            /**********************************************************************
             * Deposit
             * ---------------------------------------------------------------------
             * Adds the amount into the account
             **********************************************************************
            virtual void Deposit(Date today, float amount);

            /**********************************************************************
             * Withdrawal
             * ---------------------------------------------------------------------
             * Deducts amount from account unless there are insufficient funds
             * - returns FALSE if withdrawal can't be perform
             **********************************************************************/
            virtual bool Withdrawal(Date today, float amount);
```

Account Class

```
            /*****************************************************************
             * TransferFunds
             * ---------------------------------------------------------------
             * Transfers amount from the transferAcct to this account
             * (Withdrawal from transferAcct - Deposit to this Acct
             * - returns FALSE if the transfer can't be performed
             *   either due to invalid account options or insuffcient
             *   funds in the transferAcct
             * - returns TRUE if transfer is completed
             *****************************************************************/
            virtual bool TransferFunds(Date today, Account *transferAcct, float amount);

            /*****************************************************************
             * Update Acct -
             * ---------------------------------------------------------------
             * Updates interest and overdraft charges
             *       based on the difference in months between the
             *       lastAccessDate and today
             * This should be called with each transaction
             *****************************************************************/
            virtual void UpdateAcct(Date today);

            /*****************************************************
                ACCESSORS
             *****************************************************/
            Date    GetOpenDate()       const;
            string  GetName()           const;
            int     GetAcctNum()       const;
            float   GetBalance ()      const;
            Date    GetLastTransDate() const;

 protected:
            string        clientName;
            int           acctNumber;
            float         currentBalance;
            Date          openDate;
            Date          lastAccessDate;

 };
```

Checking Class

```
class Checking: public Account
{

public:
            /*****************************************************
                 CONSTRUCTORS & DESTRUCTOR
             *****************************************************/
            Checking();
            Checking(Date openingDate, string name, int accountNum, float balance,
                     float overdraftFee);
            virtual ~Checking();

            /*****************************************************
                 MUTATORS
             *****************************************************/
            void SetAllValues(Date openingDate, string name, int accountNum, float balance,
                              float  overdraftFee);
            void SetOverdraftFee(float overdraftFee);
            virtual bool Withdrawal(Date today, float amount);
            virtual void UpdateAcct(Date today);

            /*****************************************************
                ACCESSORS
             *****************************************************/
            float GetOverdraftFee() const;

Protected:
            float overdraftPenalty;

};
```

Savings Class

```
class Savings: public Account
{

public:
            /*************************************************
                 CONSTRUCTORS & DESTRUCTOR
             *************************************************/
            Savings();
            Savings(Date openingDate, string name, int accountNum, float balance,
                    float intRate);
            virtual ~Savings();

            /*************************************************
                 MUTATORS
             *************************************************/
            void SetAllValues(Date openingDate, string name, int accountNum, float balance,
                              float intRate);
            void SetInterestRate(float intRate);
            virtual bool Withdrawal(Date today, float amount);
            virtual void UpdateAcct(Date today);

            /*************************************************
                ACCESSORS
             *************************************************/
            float GetInterestRate() const;

protected:
            float  interestRate;

};
```

MoneyMarket Class

```
class MoneyMarket: public Savings
{

public:
            /*************************************************
                 CONSTRUCTORS & DESTRUCTOR
             *************************************************/
            MoneyMarket();
            MoneyMarket(Date openingDate, string name, int accountNum, float balance,
                        float intRate, float withdrPenalty);
            virtual ~MoneyMarket();

            /*************************************************
                 MUTATORS
             *************************************************/
            void SetAllValues(Date openingDate, string name, int accountNum, float balance,
                              float intRate, float withdrPenalty);
            void SetWithdrawalPenalty(float withdrPenalty);
            virtual bool Withdrawal(Date today, float amount);

            /*************************************************
                ACCESSORS
             *************************************************/
            float GetWithdrawalPenalty() const;

protected:
            float withdrawalPenalty;

};
```