COMP 2710

Software Construction

Lab 4

Robinson Davis

November 19, 2014

1 Analysis

This program will be a simple bank transaction application. It will allow users to perform multiple functions. They are as listed:

1. Create a bank account of type:
   a. Savings
   b. Checking
   c. Money Market
   d. CD account
2. Perform transactions:
   a. Withdrawal
   b. Deposit
   c. Transfer
   d. Inquiry
3. Display all recorded transactions
4. Display all deposits (with account name and types)
5. Display all withdrawals (with account name and types)
6. Display a final total Bank balance of all deposits minus withdrawals and fees

Functionality will be driven by user input, chosen through the menu below:

> (1) Create account*, (2) Deposit fund, (3) Withdraw fund, (4) Transfer fund, (5) Inquiry, (6) Print all transactions, (7) Print Bank balance, (8) Quit
>
> *(1) will be able to create any type of account.

Appropriate use cases can be found in the following section.

1.2 Use Cases

Basic Flow: Menu Option (1)

Use Case 1 - Menu Option (1)	
Description	User chooses to select menu option (1) and create an account. They may choose to create any type of account.
Assumptions	User successfully creates an account.
Actor(s)	User Aubie Bank
Steps	1. User selects choice 1. 2. Name is collected and stored.
Variation(s)	User may choose to create one of the following accounts after name is collected: 1. Savings 2. Checking 3. Money Market 4. CD Account
Precondition 1	There is a user
Precondition 2	User selects menu option (1)
Precondition 3	User Selects Valid choice
Post condition 1	An appropriate account has been created.

Alternate Flow: Create Savings

Use Case 1A - Menu Option 1 [Savings]	
Description	User chooses to create a savings account.
Assumption	A new saving account is successfully created.
Actor(s)	User Aubie Bank
Steps	1. User chooses to create savings account. 2. SavingAccount object is created. 3. Pointer to BankAccount object is successfully stored in allAccounts.
Variation(s)	None
Precondition 1	User chooses Savings
Post condition 1	Savings account created

Alternate Flow: Create Checking

Use Case 1B - Menu Option 1 [Checking]	
Description	User chooses to create a checking account.
Assumption	A new checking account is successfully created.
Actor(s)	User Aubie Bank
Steps	1. User chooses to create checking account. 2. CheckingAccount object is created. 3. Pointer to BankAccount object is successfully stored in allAccounts.
Variation(s)	None
Precondition 1	User chooses Checking
Post condition 1	Checking account created

Alternate Flow: Create Money Market

Use Case 1C - Menu Option 1 [Money Market]	
Description	User chooses to create a money market account.
Assumption	A new money market account is successfully created.
Actor(s)	User Aubie Bank
Steps	1. User chooses to create money market account. 2. MoneyMarketAccount object is created. 3. Pointer to BankAccount object is successfully stored in allAccounts.
Variation(s)	None
Precondition 1	User chooses Money Market
Post condition 1	Money Market account created

Alternate Flow: Create CD

Use Case 1D - Menu Option 1 [CD]	
Description	User chooses to create a CD account.
Assumption	A new CD account is successfully created.
Actor(s)	User Aubie Bank
Steps	1. User chooses to create CD account. 2. CDAccount object is created. 3. Pointer to BankAccount object is successfully stored in allAccounts.
Variation(s)	None
Precondition 1	User chooses CD
Post condition 1	CD account created

Basic Flow: Menu Option (2)

Use Case 2 - Menu Option (2)	
Description	User chooses to select menu option (2) and deposit funds into an account.
Assumptions	User successfully deposits funds.
Actor(s)	User Aubie Bank
Steps	1. User selects choice two 2. User enters account type 3. User enters deposit amount 4. User's funds are deposited
Variation(s)	None.
Precondition 1	There is a user
Precondition 2	User selects menu option (2)
Precondition 3	Account exists
Precondition 3	User Selects Valid choice
Post condition 1	Funds have been deposited into the appropriate account.

Basic Flow: Menu Option (3)

Use Case 3 - Menu Option (3)	
Description	User chooses to select menu option (3) and withdraw funds from an account.
Assumptions	User successfully withdraws funds.
Actor(s)	User Aubie Bank
Steps	1. User selects choice three 2. User enters account type 3. User enters withdrawal amount 4. User's funds are withdrawn
Variation(s)	User may choose to withdraw from one of the following accounts 1. Savings 2. Checking 3. Money Market 4. CD Account
Precondition 1	There is a user
Precondition 2	User selects menu option (3)
Precondition 3	User Selects Valid choice
Precondition 4	Account exists
Post condition 1	Funds have been withdrawn from the appropriate account.

Alternate Flow: Withdraw Savings

Use Case 3A - Menu Option 3 [Savings]	
Description	User chooses to withdrawal from a savings account.
Assumption	Funds are successfully withdrawn.
Actor(s)	User Aubie Bank
Steps	1. Checks balance 2. Remove funds if possible a. No restrictions 3. Notifies user if not
Variation(s)	None
Precondition 1	User chooses Savings
Precondition 2	Account exists
Post condition 1	Funds are withdrawn
Post condition 2	User is notified of insufficient funds.

Alternate Flow: Withdraw Checking

Use Case 3B - Menu Option 3 [Checking]	
Description	User chooses to withdrawal from a checking account.
Assumption	Funds are successfully withdrawn.
Actor(s)	User Aubie Bank
Steps	1. Checks balance 2. Remove funds if possible a. If balance < $500 add FEE 3. Notifies user if not
Variation(s)	None
Precondition 1	User chooses checking
Precondition 2	Account exists
Post condition 1	Funds are withdrawn
Post condition 2	User is notified of insufficient funds.

Alternate Flow: Withdraw Money Market

Use Case 3C - Menu Option 3 [Money Market]	
Description	User chooses to withdrawal from a money market account.
Assumption	Funds are successfully withdrawn.
Actor(s)	User Aubie Bank
Steps	1. Checks balance 2. Remove funds if possible a. If user has withdrawn more than twice in time frame, charge FEE 3. Notifies user if not
Variation(s)	None
Precondition 1	User chooses money market
Precondition 2	Account exists
Post condition 1	Funds are withdrawn
Post condition 2	User is notified of insufficient funds.

Alternate Flow: Withdraw CD

Use Case 3D - Menu Option 3 [CD]	
Description	User chooses to withdrawal from a CD account.
Assumption	Funds are successfully withdrawn.
Actor(s)	User Aubie Bank
Steps	1. Checks balance 2. Remove funds if possible a. Charge PENALTY 3. Notifies user if not
Variation(s)	None
Precondition 1	User chooses CD
Precondition 2	Account exists
Post condition 1	Funds are withdrawn
Post condition 2	User is notified of insufficient funds.

Basic Flow: Menu Option (4)

Use Case 4 - Menu Option (4)	
Description	User chooses to select menu option (4) and transfer funds
Assumptions	User successfully transfers funds.
Actor(s)	User Aubie Bank
Steps	1. User selects choice four 2. User enters sender account name 3. User enters sender account type 4. User enters receiver account name 5. User enters receiver account type 6. User's funds are transferred
Variation(s)	None
Precondition 1	There is a user
Precondition 2	Accounts exist
Precondition 3	User Selects Valid choice
Post condition 1	Funds have been withdrawn from the appropriate account.

Basic Flow: Menu Option (5)

Use Case 5 - Menu Option (5)	
Description	User chooses to select menu option (5) and inquiry an account.
Assumptions	User successfully inquiries an account.
Actor(s)	User Aubie Bank
Steps	1. Account name is collected 2. Account type is collected 3. Balance is displayed
Variation(s)	None
Precondition 1	There is a user
Precondition 2	Account exists
Precondition 3	User Selects valid choice
Post condition 1	Inquiry has been displayed.

Basic Flow: Menu Option (6)

Use Case 6 - Menu Option (6)	
Description	User chooses to select menu option (6) and print all transactions
Assumptions	All transactions are successfully displayed.
Actor(s)	User Aubie Bank
Steps	1. Print out each transaction in the transaction map
Variation(s)	None
Precondition 1	There is a user
Precondition 2	Transactions exists
Precondition 3	User Selects valid choice
Post condition 1	All transactions have been displayed

Basic Flow: Menu Option (7)

Use Case 6 - Menu Option (7)	
Description	User chooses to select menu option (7) and display bank balance
Assumptions	Bank balance is successfully displayed.
Actor(s)	User Aubie Bank
Steps	1. Find each bank account 2. Add all deposits 3. Subtract all withdrawals 4. Calculate bank balance 5. Display bank balance
Variation(s)	None
Precondition 1	There is a user
Precondition 2	Bank accounts exist
Precondition 3	User Selects valid choice
Post condition 1	All bank balances have been calculated
Post condition 2	All bank balances have been displayed

3 Design

There will be numerous classes to aid in the functionality of this program. They are defined as follows:

***Please note: All classes will have accessor and set functions for all variables, as well as default and copy constructors. These will not be listed as they are self-explanatory. However, if another constructor is created to take parameters, it will be listed.**

1. **Global class -** Used to maintain the main() function.

   a. **Variables**

      i. None.

   b. **Functions**

      i. **int** main()
         1. Contains, a vector of pointers to BankAccount objects where all accounts in the vector have current withdrawal or deposit activities.
         2. Runs the program.
         3. Will handle all code for this program, as a large majority of necessary functionality is already provided in corresponding classes.

2. **BankAccount class** – Used to maintain information common across all account types in the system.

   a. Variables

      i. **private** string name
         1. Name of the bank account's owner.

      ii. **private** string accountType
         1. Type of bank account

      iii. **private** double balance
         1. Balance in the bank account.

      iv. **protected** map<string, vector<double> transactions
         1. A record of all transactions in the account.
         2. Uses "Deposit" and "Withdrawal" as keys
      v. **protected** vector<BankAccount*> allAccounts
         1. Contains pointers to all valid BankAccount objects

b. Functions

i. **public** BankAccount(string newName, string newAccountType, double newBalance)
   1. Sets name = newName
   2. Sets accountType = newAccountType
   3. Sets balance = newBalance
   4. Sets transactions = NULL

ii. **public void** deposit()
   1. If amount is non-negative, amount will be added to balance.
   2. Records results in transactions map.

iii. **virtual public void** withdrawal()
   1. If amount is non-negative and is less than or equal to balance, amount will be subtracted from balance.
      a. If amount is greater than balance, user will be informed.

iv. **virtual public void** printTransactions()
   1. Displays information for all transactions in the transactions map.
   2. Virtual so that all derived classes can properly display transaction information.

v. **virtual public void** computeBankBalance()
   1. Depending on type of account, returns the amount of deposit, withdrawal, fees and penalties of each transaction in an account.
   2. Virtual so that each account can properly retrieve transaction information.

3. **SavingAccount class (Derived from BankAccount) -** Used to maintain information about saving accounts.

a. Variables

i. None.

b. **Functions**

i. **public** SavingAccount(string newName, string newAccountType, double newBalance)
   1. Sets name = newName
   2. Sets accountType = newAccountType
   3. Sets balance = newBalance
   4. Sets transactions = NULL

ii. **public void** withdrawal()
   1. If amount is non-negative and is less than or equal to balance, amount will be subtracted from balance.
      a. If amount is greater than balance, user will be informed.
   2. User can withdraw any amount of funds without any restriction.
   3. Records results in transactions map.

iii. **public void** printTransactions()
   1. Displays information for all SavingAccount transactions in the transactions map.

iv. **public void** computeBankBalance()
   1. Returns the amount of all deposit and withdrawal transactions in an account.

4. **CheckingAccount class (Derived from BankAccount)** - Used to maintain information about checking accounts.

   a. **Variables**

      i. **const double** FEE = 2.50
         1. Fee for if a user withdrawals an amount which causes the checking account balance to fall under $500.00.

   b. **Functions**

      i. **public** CheckingAccount(string newName, string newAccountType,
                                    double newBalance)
         1. Sets name = newName
         2. Sets accountType = newAccountType
         3. Sets balance = newBalance
         4. Sets transactions = NULL

      ii. **public void** withdrawal()
         1. If amount is non-negative and is less than or equal to balance, amount will be subtracted from balance.
            a. If amount is greater than balance, user will be informed.
         2. If amount is non-negative and is less than or equal to balance, **AND** if the balance of the account after the withdrawal is less than $500.00 then (amount + FEE) will be subtracted from the balance.
         3. Records results in transactions map.

      iii. **public void** printTransactions()
         1. Displays information for all CheckingAccount transactions in the transactions map.

      iv. **public void** computeBankBalance()
         1. Returns the amount of all deposit and withdrawal transactions in an account.

5. **MoneyMarketAccount class (Derived from BankAccount) -** Used to maintain information about money market accounts.
   a. **Variables**

      i. **const double** FEE = 1.50
         1. Fee for if a user withdrawals more than two times in a determined time.

      ii. **int** numOfWithdrawals
         1. Records the number of withdrawals on the account.

b. **Functions**

   i. **public** MoneyMarketAccount(string newName, string newAccountType,
      double newBalance)
      1. Sets name = newName
      2. Sets accountType = newAccountType
      3. Sets balance = newBalance
      4. Sets transactions = NULL
      5. Sets numOfWithdrawals = 0

   ii. **public void** withdrawal(double amount)
      1. If amount is non-negative and is less than or equal to balance, amount will be subtracted from balance.
         a. If amount is greater than balance, user will be informed.
      2. If amount is non-negative and is less than or equal to balance, **AND** if the user has withdrawn more than twice (indicated by numOfWithdrawals) in a given time, then (amount + FEE) will be subtracted from balance.
      3. Records results in transactions map.

   iii. **public void** printTransactions()
      1. Displays information for all MoneyMarketAccount transactions in the transactions map.

   iv. **public void** computeBankBalance()
      1. Returns the amount of all deposit and withdrawal transactions in an account.

6. **CDAccount class (Derived from BankAccount) -** Used to maintain information about CD accounts.

a. **Variables**

   i. **int** interestRate
      1. Interest rate on the CD account.

   ii. **int** numOfDeposits
      1. The number of deposits made by the account. For ever two deposits, an amount of (interestRate + balance) will be added to the account.
      2. Otherwise, the penalty would be impossible to calculate.

   iii. **double** totalInterest
      1. Keeps track of total interest earned so that PENALTY can be calculated.

   iv. **double** PENALTY = (0.25 * totalInterest)
      1. Penalty if user withdrawals.

b. **Functions**

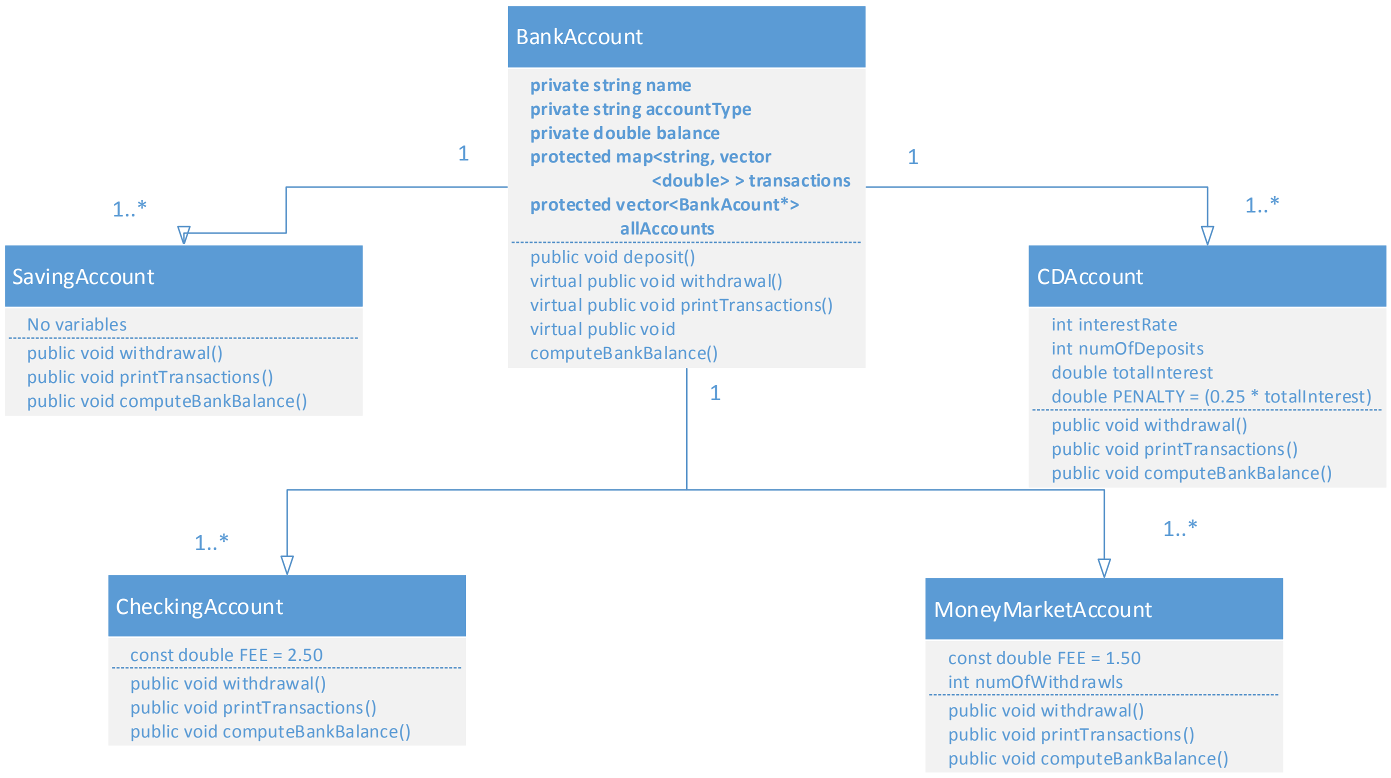
i. **public** CDAccount(string newName, string newAccountType,
double newBalance, int newInterestRate)
   1. Sets name = newName
   2. Sets accountType = newAccountType
   3. Sets balance = newBalance
   4. Sets transactions = NULL
   5. Sets interestRate = newInterestRate
   6. Sets numOfDeposits = 0
   7. Sets totalInterest = 0

ii. **public void** withdrawal(double amount)
   1. If amount is non-negative and is less than or equal to balance, (amount + PENALTY) will be subtracted from balance.
      a. If amount is greater than balance, user will be informed.
   2. Records results in transactions map.

iii. **public void** printTransactions()
   1. Displays information for all MoneyMarketAccount transactions in the transactions map.

iv. **public void** computeBankBalance()
   1. Returns the amount of all deposit and withdrawal transactions in an account.

2.1 Class Diagram


BankAccount
private string name
private string accountType
private double balance
protected map<string, vector <double> > transactions
protected vector<BankAcount*> allAccounts
public void deposit()
virtual public void withdrawal()
virtual public void printTransactions()
virtual public void computeBankBalance()
1
1..*
SavingAccount
No variables
public void withdrawal()
public void printTransactions()
public void computeBankBalance()
1
1..*
CDAccount
int interestRate
int numOfDeposits
double totalInterest
double PENALTY = (0.25 * totalInterest)
public void withdrawal()
public void printTransactions()
public void computeBankBalance()
1
1..*
CheckingAccount
const double FEE = 2.50
public void withdrawal()
public void printTransactions()
public void computeBankBalance()
1..*
MoneyMarketAccount
const double FEE = 1.50
int numOfWithdrawls
public void withdrawal()
public void printTransactions()
public void computeBankBalance()

2.2 Data Flow Diagram

3 Tests

Tests are performed by close monitoring of the program in Microsoft Visual 2013. All program function and variables are monitored in debugging using appropriate breakpoints and proper recognition of errors.

1. Ensure that proper header and footers display.
2. Ensure menu displays.
3. Ensure user can pick a choice.
4. Ensure choice is valid.
5. Ensure that all objects are successfully created, and all data properly stored into them.