CSE2122 (Spring 2020)
Homework 6, due 4/17/2020 11:59pm

Before you start, please login your stdlinux account and download the solution file and the template program from the class directory to your stdlinux directory. You can use the command "`cp /class/cse2122/hw6/* ./.`". These files are also available on Carmen.

Submit your code to the designated dropbox on Carmen under the Assignments tab. Late submissions receive a 10% penalty for each day. No late submissions after the second day.

Skills needed to complete this assignment: classes and inheritance.

Banks have many different types of accounts often with different rules for fees associated with transactions such as withdrawals. Customers can transfer funds between accounts incurring the appropriate fees associated with withdrawal of funds from one account.

Write a program with a base class for a bank account and two derived classes as described below representing accounts with different rules for withdrawing funds. Also write a function that transfers funds from one account of any type to another account of any type.

A transfer is a withdrawal from one account and a deposit into the other.

Since the transfer can be done at any time with any type of accounts, the `withdraw` function in the classes must be virtual. The transfer function utilizes polymorphism in order to transfer funds between any subclasses of `BankAccount`. So, the transfer function should have pass-by-reference `BankAccount` parameters as the "from" and "to" bank accounts. The transfer function returns a Boolean value, `true` if this transfer is finished successfully, `false` otherwise (for example, non-positive transfer amounts or unsuccessful withdrawal).

Create a base class called `BankAccount` that has the name of the owner of the account (a string variable) and the balance in the account (a double variable) as data members. The default name and balance are set to `Joe Doe` and `0`. Include member functions `deposit` and `withdraw` (each with a double for the amount as an argument) and accessor functions `getName` and `getBalance`. Function `deposit` will add the amount to the balance and `withdraw` will subtract the amount from the balance.

Also create a class called `MoneyMarketAccount` that is derived from `BankAccount`. In a `MoneyMarketAccount`, the user gets 2 free successful withdrawals. After the free withdrawals have been used, a withdrawal fee of $1.50 is deducted from the balance per successful withdrawal. Hence, the class must have a data member to keep track of the number of withdrawals. It also must override the `withdraw` definition.

Finally, create a `CDAccount` class (to model a Certificate of Deposit) derived from `BankAccount` which in addition to having the name and balance, also has an interest rate (`0.02` by default). CDs incur penalties for early withdrawal of funds. Assume that a withdrawal of funds (any amount) incurs a penalty of 25% of the annual interest earned on the account (just assume that the annual interest is equal to the interest rate times the balance before withdrawal). Assume the amount withdrawn plus the penalty are deduced from the account balance. Again, it must override the `withdraw` definition.

For all three classes, the `withdraw` function should return a Boolean value indicating the status (`false` if amount is non-positive or insufficient funds for the withdrawal, after adding the fees, to take place). The `deposit` function should return a Boolean value as well; deposit should return `false` if the amount to deposit is not positive. If `withdraw` or `deposit` returns `false`, then balance must remain unchanged. Both functions return `true` if successful. The point of the Boolean returned values is to indicate success or failure, because these functions may refuse to withdraw or deposit if conditions are not right. For the purposes of this assignment, you do not need to implement other functions or properties of these accounts.

Figure 1 shows the hierarchy of the classes.

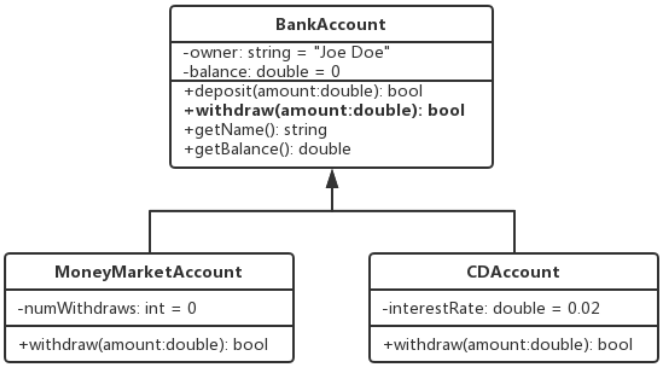

Figure 1 Hierarchy of classes derived from BankAccount class. A '-' sign indicates private or protected member and a '+' sign indicates public members. Virtual functions are boldfaced. The type of variables and return type of functions are separated with a colon ':'.

Note that the member variable `balance` is NOT publicly accessible outside the class, but we want it to be accessible in the derived classes. In addition to the functions in Figure 1, all three classes must have constructors that initialize their members (either from the input parameters or a constant value). You can get a hint by looking at the constructors used in the example application program. No other public functions must be defined in these classes, other than constructors and those seen in Figure 1. You can define private helper functions if you want. Don't forget to define `transfer` function in the application program, which is not a member of any class.

You should write/complete and submit seven different files (see sections *Setting up the programming environment* and *Submit* below), one header file (.h) and one implementation file (.cpp) for each class, and an application program. The header files and implementation files for the three classes must be named *bank_account.h, back_account.cpp, money_marketing_account.h, money_marketing_account.cpp, cd_account.h, cd_account.cpp*, respectively.

Input/Output Example

The following examples and the solution file are based on the example application program. User input is underlined.

```
Enter Money Marketing Account owner's name: David
Enter CD Account owner's name: Monica
Enter CD Account's interest rate (0 to 1): 0.4

Enter amount to deposit to David's account: 100

Successful deposit.
David's account balance: 100.00

Enter amount to deposit to Monica's account: 1000

Successful deposit.
Monica's account balance: 1000.00

Enter amount to withdraw from David's account: 0

Unsuccessful withdraw. Insufficient funds (100.00) or invalid amount
(0.00)
David's account balance: 100.00

Enter amount to withdraw from David's account: 10

Successful withdraw.
David's account balance: 90.00

Enter amount to withdraw from Monica's account: -10

Unsuccessful withdraw. Insufficient funds (1000.00) or invalid amount
(-10.00)
Monica's account balance: 1000.00

Enter amount to withdraw from Monica's account: 10

Successful withdraw.
Monica's account balance: 890.00

Enter amount to transfer from Monica's account to David's account: 900

Unsuccessful transfer. Insufficient funds (890.00) or invalid amount
(900.00)
Monica's account balance: 890.00
David's account balance: 90.00

Enter amount to transfer from David's account to Monica's account: 5

Successful transfer.
David's account balance: 85.00
Monica's account balance: 895.00
```

```
Enter amount to withdraw from David's account: 5

Successful withdraw.
David's account balance: 78.50
```

Program Compilation and Testing

In this assignment, the project contains multiple files. For each class, you need to write the class header in a header file (.h), and the implementation in a C++ file (.cpp).

Testing each function/class before moving to the next one is usually a good idea. However, one class (one header file and one implementation file) cannot be individually compiled and tested without a main function. You may need to write your own test main program. Following is an example test program (cd_test.cpp) for the class `CDAccount`.

```
#include <iostream>
#include "bank_account.h"
#include "cd_account.h"

using namespace std;

int main(){
  CDAccount cdtest;
  cout << cdtest.getName() << endl;
  cout << cdtest.getBalance() << endl;
  cout << cdtest.withdraw(100) << endl;
  /* more test cases… */
  return 0;
}
```

To run the test program, you need to compile the class files and the test program together. For example, to compile the above test program, you can use the following command. You do not need to compile header files, but the class header in the header file must match your implementations.

```
g++ cd_test.cpp bank_account.cpp cd_account.cpp -std=c++11 –pedantic-errors
```

You can use the command

```
g++ *.cpp -std=c++11 –pedantic-errors
```

to compile all C++ files under the current directory.

Program Submission

You must submit only one zip file containing your six class files and the application program. In terminal go to the folder that has all your files in it and type the following command (in one line):

```
zip hw6.zip hw6_main.cpp bank_account.h bank_account.cpp
money_market_account.h money_market_account.cpp cd_account.h
cd_account.cpp
```

You can also use file compression softwires to compress them into one zip file.

Each program in your submission must have the following at the top:

```
/*
File name: (your file name)
Created by: (your name)
Created on: (date)
Synopsis: (what your program does)
*/
```

Make sure your code is formatted properly. Your program should have appropriate indention and necessary comments. See textbook on section 2.5 or any code example in textbook/lecture slides.

Submit your zip file to the designated dropbox on Carmen. You can find the dropbox under the Assignments tab.