

Homework#3

CS331 Introduction to C++ & Object-Oriented Programming, Summer 2018

Due: July 27 (Friday)

Instruction:

- This homework has five programming assignments
- Make one file (YourLastName_YourFirstName_CS331_HW3.zip) for your submission, and submit it to Blackboard.
- The zip file will have Q1, Q2, Q3, Q4 and Q5 directories for your source code of each question.
- The directory will include a root directory which contains all the required source code for the assignment including resources (e.g., Visual Studio Project file) but excluding debug directory.
- **Your source codes should be properly indented and documented to have professional appearance. You should also provide proper comments for the program and variables.**
- **The evaluation is based on correct implementation and execution.**

Q1. After defining the classes in the class hierarchy described below, write a program that creates objects of each class and test their member functions.

All customers at this bank can deposit (i.e., credit) money into their accounts, and withdraw (i.e., debit) money from their accounts. More specific types of accounts also exist. Savings accounts, for instance, earn interest on the money they hold. Checking accounts, on the other hand, charge a fee per transaction (i.e., credit or debit). The inheritance hierarchy contains base-class `Account`, and derived classes `SavingsAccount` and `CheckingAccount` that inherit from class `Account`.

Design:

The base class: `Account`

File names: `Account.h`, `Account.cpp`

Base-class Account	
Data Members	Description
<code>double balance;</code>	A private member <i>balance</i> represents the account balance.

Member Functions	Description
<code>Account(double);</code>	A constructor that receives an initial balance and uses it to initialize the data member. The constructor should validate the initial balance to <u>ensure that it's greater than or equal to 0.0</u> . If not the balance should be set to 0.0 and the constructor should display an error message, indicating that the initial balance was invalid. (<i>Hint: throw <code>invalid_argument</code></i>)
<code>void credit(double);</code>	A public member function <code>credit</code> adds an amount to the account current balance.
<code>bool debit(double);</code>	A public member function <code>debit</code> subtracts an amount from the account balance. It should <u>ensure that the debit amount does not exceed the Account's balance</u> . If it does, the balance should be left unchanged and the function should the message " <i>Debit amount exceeded account balance</i> ".
<code>double getBalance();</code>	A public member function <code>getBalance</code> returns the account balance.
<code>void setBalance(double);</code>	A public member function <code>setBalance</code> sets the account balance.

A derived class: **SavingsAccount**

File names: SavingsAccount.h, SavingsAccount.cpp

Derived-class SavingsAccount	
Data Members	Description
<code>double interestRate;</code>	Derived-class SavingsAccount should inherit the functionality of an Account , but also include a private data member of type <code>double</code> indicating the <i>interest rate</i> (percentage) assigned to the Account .
Member Functions	Description
<code>SavingsAccount(double, double);</code>	SavingsAccount 's constructor should receive the initial balance, as well as an initial value for the SavingsAccount 's interest rate. The constructor does a job similar with the base class construction. <u>For initialing the balance, use the base class's constructor</u> . The constructor should also validate the initial interest rate to ensure that it's greater than or equal to 0.0. If not the rate should be set to 0.0 and the constructor should display an error message, indicating that the initial rate was invalid.
<code>double calculateInterest();</code>	A public member function <code>calculateInterest</code> returns a <code>double</code> indicating the amount of interest earned by an account. It should determine this amount by multiplying the interest rate by the account balance.

Note: **SavingsAccount** should inherit member functions `credit` and `debit` as is without redefining them.

A derived class: **CheckingAccount**

File names: **CheckingAccount.h**, **CheckingAccount.cpp**

Derived-class CheckingAccount	
Data Members	Description
<code>double transactionFee;</code>	Derived-class CheckingAccount should inherit from base-class Account and include an additional private data member of type double that represents the <i>fee</i> charged per transaction.
Member Functions	Description
<code>CheckingAccount(double, double);</code>	CheckingAccount 's constructor should receive the initial balance, as well as a parameter indicating a fee amount. The constructor does a job similar with the base class construction. <u>For initializing the balance, use the base class's constructor.</u> The constructor should also validate the initial transaction fee to ensure that it's greater than or equal to 0.0. If not the transaction fee should be set to 0.0 and the constructor should display an error message, indicating that the initial fee was invalid.
<code>void credit(double);</code>	Class CheckingAccount should refine member functions credit of Account . This public function adds an amount to the account balance, and then charges fee. CheckingAccount 's credit always charge fee.
<code>bool debit(double);</code>	Class CheckingAccount should refine member functions debit of Account . This public function subtracts an amount to the account balance, and then should charge fee <u>only if money is actually withdrawn</u> (i.e., the debit amount does not exceed the account balance). [<i>Hint</i> : Define Account 's debit function so that it returns a bool indicating whether money was withdrawn. Then use the return value to determine whether a fee should be charged.]
Helper Member Function	Description
<code>void chargeFee();</code>	A private member function to charge fee. This function subtract the transaction fee from the account balance.

Given: HW3Q1.cpp.

To do:

- 1) Complete **Account.h**, **Account.cpp**, **SavingsAccount.h**, **SavingsAccount.cpp**, **CheckingAccount.h**, and **CheckingAccount.cpp**
- 2) Test with the driver program **HW3Q1.cpp**.

Running Result:

```
account1 balance: $50.00
account2 balance: $25.00
account3 balance: $80.00

Attempting to debit $25.00 from account1.

Attempting to debit $30.00 from account2.
Debit amount exceeded account balance.

Attempting to debit $40.00 from account3.
$1.00 transaction fee charged.

account1 balance: $25.00
account2 balance: $25.00
account3 balance: $39.00

Crediting $40.00 to account1.

Crediting $65.00 to account2.

Crediting $20.00 to account3.
$1.00 transaction fee charged.

account1 balance: $65.00
account2 balance: $90.00
account3 balance: $58.00

Adding $2.70 interest to account2.

New account2 balance: $92.70
Press any key to continue . . .
```

Q2. (20 points) Develop a polymorphic banking program (named HW3Q1.cpp) using the Account hierarchy create in Part I.

In HW3Q2.cpp,

- 1) Create a vector of Account pointers to SavingsAccount and ChekingAccount objects.
- 2) Initialize vector with Accounts like following:

```
accounts[0] = new SavingsAccount{25.0, .03};
accounts[1] = new CheckingAccount{80.0, 1.0};
accounts[2] = new SavingsAccount{200.0, .015};
accounts[3] = new CheckingAccount{400.0, .5};
```

- 3) For each Account in the vector,
 - 3.1) First, display the account balance
 - 3.2) Allow the user to specify an amount of money to withdraw from the account using member function `debit`
 - 3.3) Allow the user to specify an amount of money to deposit into the account using member function `credit`.
 - 3.4) To know whether an account is a SavingsAccount, use downcast pointer, .e.g,
`SavingsAccount* savingsAccountPtr = dynamic_cast<SavingsAccount*>(accounts[i]);`

3.5) If an account is a `SavingsAccount` (i.e., `savingsAccountPtr != 0`), calculate the amount of interest owed to the Account using member function `calculateInterest`, then add the interest to the account balance using member function `credit`.

3.6) After processing an Account, print the updated account balance obtained by invoking base-class member function `getBalance`.

Given: HW3Q2.cpp.

To do:

- 1) Complete HW3Q2.cpp
- 2) Test HW3Q2.cpp with Account.h, Account.cpp, SavingsAccount.h, SavingsAccount.cpp, CheckingAccount.h and CheckingAccount.cpp you developed in Q1.

Running example:

```
Account 1 balance: $25.00
Enter an amount to withdraw from Account 1: 10
Enter an amount to deposit into Account 1: 100
Adding $3.45 interest to Account 1 (a SavingsAccount)
Updated Account 1 balance: $118.45

Account 2 balance: $80.00
Enter an amount to withdraw from Account 2: 50
$1.00 transaction fee charged.
Enter an amount to deposit into Account 2: 20
$1.00 transaction fee charged.
Updated Account 2 balance: $48.00

Account 3 balance: $200.00
Enter an amount to withdraw from Account 3: 200
Enter an amount to deposit into Account 3: 500
Adding $7.50 interest to Account 3 (a SavingsAccount)
Updated Account 3 balance: $507.50

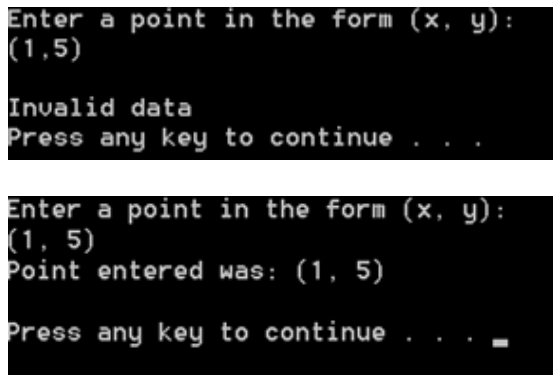
Account 4 balance: $400.00
Enter an amount to withdraw from Account 4:
```

Q3. Write a program that accomplishes the following:

- 1) Create a user-defined class *Point* that contains the private integer data members *xCoordinate* and *yCoordinate*
- 2) Declares stream insertion and stream extraction overloaded operator functions as *friends* of the class.
- 3) Define the stream insertion and stream extraction operator functions.
 - The stream extraction operator function should determine whether the data entered is valid, and, if not, it should set the *failbit* to indicate improper input. The valid input format is (x, y). Refer to running examples below.
 - The stream insertion operator should not be able to display the point after an input error occurred.

- 4) Test your implementation with a given test program (HW3Q3.cpp). The main function of the test program tests input and output of user-defined class `Point`, using the overloaded stream extraction and stream insertion operators.

Running Examples:



```
Enter a point in the form (x, y):
(1,5)

Invalid data
Press any key to continue . . .

Enter a point in the form (x, y):
(1, 5)
Point entered was: (1, 5)

Press any key to continue . . .
```

Given: HW3Q3.cpp

To do:

- 1) Complete `Point.h` (and `Point.cpp` if you include the overloaded operator functions in a separate file.)
- 2) Show your programming is correctly running.

Q4. Write a program that uses a structure to store the following inventory information in a file:

- Item description
- Quantity on hand
- Wholesale cost
- Retail cost
- Data added to inventory

The inventory record is designed with a `struct` structure like:

```
struct Invtry
{
    char desc[31];
    int qty;
    double wholeSale;
    double retail;
    char date[11];
};
```

The program should have a menu that allows the user to perform the following tasks:

- ```
1. Add a new record
2. View an existing record
3. Change an existing record
4. Exit
```

```
Enter your choice (1-4):
```

In the insertion function, `addRecord()`, quantity, cost and price should be a positive value.

- ```
1. Add a new record
2. View an existing record
3. Change an existing record
4. Exit
```

```
Enter your choice (1-4): 1
```

```
Enter the following inventory data:
```

```
Description: Jabra Elite Alexa
```

```
Quantity: 1
```

```
Wholesale cost: 169.99
```

```
Retail price: 199.99
```

```
Date added to inventory: 07/10/2018_
```

In the query function, `viewRecord()`, user enters the record number of the item. The number implies the record order in the file. 0 indicates the first record in the file, 1 is the second record, 2 is the third record, and so on. The query function displays a queried record like following:

The change function, `changeRecord()` receives a record number of the item changed and display the current information of the item. Then it receives new information for the change from user and replaces the current record with the new record.

- ```
1. Add a new record
2. View an existing record
3. Change an existing record
4. Exit
```

```
Enter your choice (1-4): 3
```

```
Enter the record number of the item: 1
```

```
Current record contents:
```

```
Description: Comply SmartCore Sport
```

```
Quantity: 10
```

```
Wholesale cost: 14.99
```

```
Retail price: 19.99
```

```
Enter the new data:
```

```
Description: Comply SmartCore Sport
```

```
Quantity: 9
```

```
Wholesale cost: 14.99
```

```
Retail price: 14.99
```

```
Date added to inventory: 07/03/2018
```

**Given:** HW3Q4.cpp

**To do:**

- 1) Complete HW3Q4.cpp
- 2) Test HW3Q4 program.

**Q5.** Implement a function

`T accum(vector <T> v)`

that forms and returns the “sum” of all items in the vector `v` passed to it.

For example, if `T` is a numeric type such as `int` or `double`, the numeric sum will be returned, and if `T` represents the STL `string` type, then the result of concatenation is returned.

Write a program to test this function with 1) getting 3 numbers from user and 2) getting 3 strings from user.

**Given:** HW3Q5.cpp

**To do:** Complete HW3Q5.cpp.

Show your programming is correctly running.