COMSC 200

Summer 2024

Programming Assignment 11

Worth 12.5 points (1.25% of your grade)

DUE: Saturday, 7/13/24 by 11:59 P.M. on Canvas

Late Pass Deadline: Tuesday, 7/16/24 by 11:59 P.M. on Canvas

This Programming Assignment directly builds off of Programming Assignment 10 with the Account, SavingsAccount, and CheckingAccount classes. Use your solution for Programming Assignment 10 as the starting point for this programming assignment.

Your solution should consist of **eight (8) files**:

Account.h

Account.cpp

SavingsAccount.h

SavingsAccount.cpp

CheckingAccount.h

CheckingAccount.cpp

The application program (since you will be modifying this for the overloaded operators)

Firstname_lastname_200_assign11.pdf (sample runs)

Please continue to use the same **naming convention** as before, where each filename should contain both your first name and your last name. If your first name is "James" and your last name is "Smith", then your Account header file should be named James_Smith_Account.h. The other six files should use the same naming convention.

Comments – worth 1.25 points (10%) of your programming assignment grade:

Your program should have at least **ten (10)** different detailed comments explaining the different parts of your program. Each individual comment should be, at a minimum, a short sentence explaining a particular part of your code. You should make each comment as detailed as necessary to fully explain your code. You should also number each of your comments (i.e., comment 1, comment 2, etc.).

Sample Runs – worth 1.25 points (10%) of your programming assignment grade:

You should submit screenshots of at least **five (5)** different sample runs of your program. To generate each sample run, download the application program file from canvas and change all of the following: (1) the values used in all of the constructor calls (when the objects are created), (2) the values used in all of the calls to the debit function, and (3) the values used in all of the calls to the credit function. Each sample run needs to use different inputs for all of these values, and your sample runs should **NOT** be the same as the sample run that is used in this write-up for the assignment. You should also number each of your sample runs (i.e., sample run 1, sample run 2, etc.). **You need to also add statements in the application program that invoke each of your overloaded operator functions – your screenshots should show the values used along with the corresponding output on the console screen**.

For this programming assignment you will be overloading the following operators in **all three classes** (Account, SavingsAccount, and CheckingAccount):

1) <<
2) +=
3) – =
4) +
5) –
6) =

1) For <<, the idea is that you want to be able to print out the balance (**without** having to call the getBalance member function) as follows:

```
Account a;
SavingsAccount s;
CheckingAccount c;

cout << a << endl; // prints the balance for the Account object a
cout << s << endl; // prints the balance for the SavingsAccount object s
cout << c << endl; // prints the balance for the CheckingAccount object c
```

2) For +=, the idea is that you want to be able to add/credit money to the account (**without** having to call the credit member function as follows):

```
Account a;
SavingsAccount s;
CheckingAccount c;

a+=50.0;  //credit's (adds) $50.00 to the balance of Account object a
```

s+=27.5; //credit's (adds) $27.50 to the balance of SavingsAccount object s

c+=35.73; //credit's (adds) $35.73 to the balance of CheckingAccount object c

3) For -=, the idea is that you want to be able to subtract/debit money from the account (**without** having to call the debit member function as follows):

Account a;

SavingsAccount s;

CheckingAccount c;

a-=50.0; //debit's (subtracts) $50.00 from the balance of Account object a

s-=27.5; // debit's (subtracts) $27.50 from the balance of SavingsAccount object s

c-=35.73; //debit's (subtracts) $35.73 from the balance of CheckingAccount object c

4) And 6) For + and =, the idea is that in addition to += (2), you should also be able to add/credit to the account as follows:

Account a, a2;

SavingsAccount s, s2;

CheckingAccount c, c2;

a=a+50.0; OR a=50.0+a; (BOTH orderings should be supported)

s=s+50.0; OR s=50.0+s; (BOTH orderings should be supported)

c=c+50.0; OR c=50.0+c; (BOTH orderings should be supported)

Note that = can also be used for statements such as a=a2;, s=s2;, and c=c2;

5) And 6) For - and =, the idea is that in addition to -= (3), you should also be able to subtract/debit from the account as follows:

Account a, a2;

SavingsAccount s, s2;

CheckingAccount c, c2;

a=a-50.0; s=s50.0;

c=c-50.0;

Note that = can also be used for statements such as a=a2;, s=s2;, and c=c2;

Note that there are a total of six (6) operators that you need to overload (<<, +=, -=, +, -, =), but you need two different versions for the + operator, meaning that you need seven (7) overloaded operators for each class. Each of the three classes needs all seven (7) overloaded operators, **which means you will need to have a total of twenty-one (21) overloaded operators.**

HINT: Balance is not a direct member of the derived classes SavingsAccount and CheckingAccount. It is a member of the base class Account, and the derived classes SavingsAccount and CheckingAccount inherit it. This means in the overloaded operators for Savings/Checking, you cannot directly access Balance, since Balance is not a direct member of either of those classes. You need to think about how to handle this.