COMSC 200

Summer 2024

Programming Assignment 10

Worth 12.5 points (1.25% of your grade)

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

Late Pass Deadline: Sunday, 7/14/24 by 11:59 P.M. on Canvas

You need to start by downloading the following file from the Programming Assignment 10 folder on Canvas: Account_app.cpp

Your solution should consist of **seven (7) files**:

Account.h

Account.cpp

SavingsAccount.h

SavingsAccount.cpp

CheckingAccount.h

CheckingAccount.cpp

Firstname_lastname_200_assign9.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.

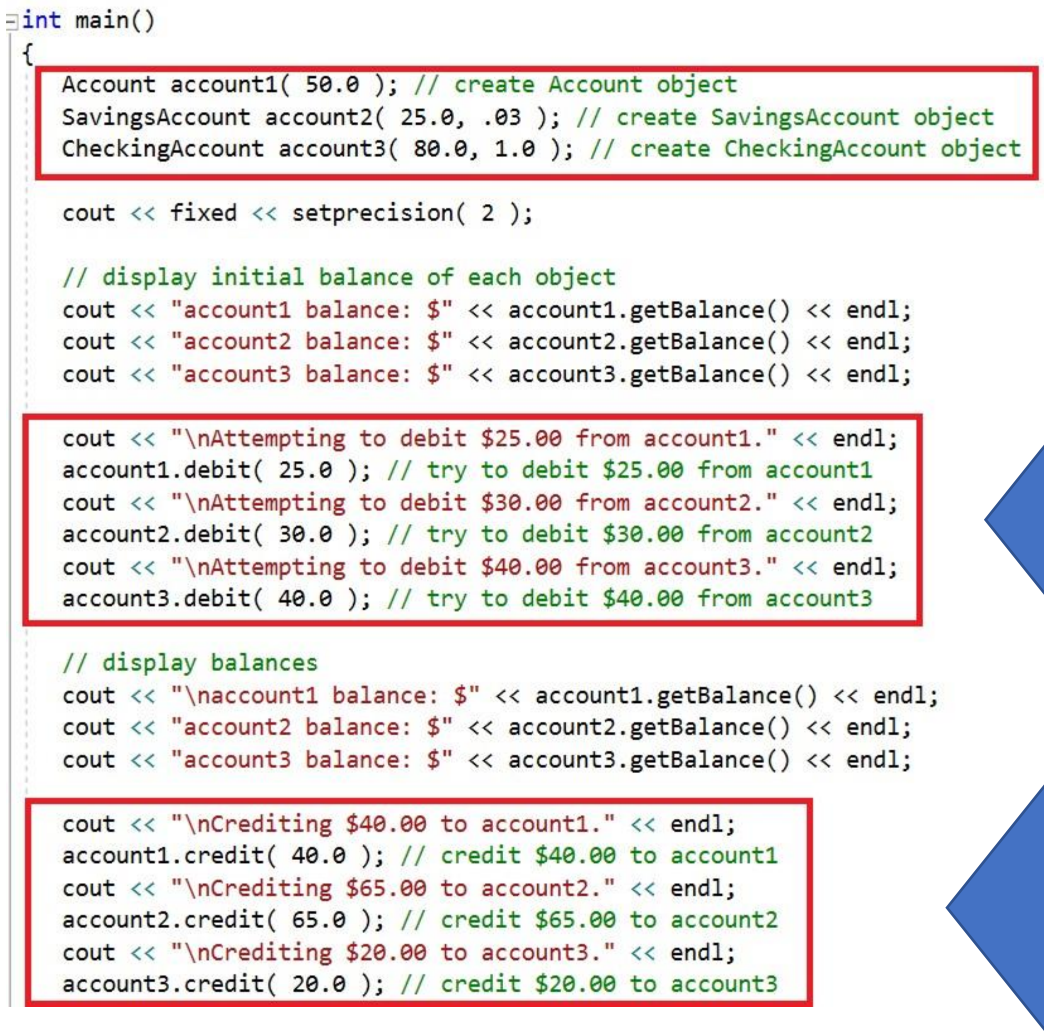
Comments – worth 1.25 points (10%) of your programming assignment grade:

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.). All of your sample runs should follow this format – for each individual sample run, screenshot (1) the values used in the constructor/debit/credit function calls and (2) the corresponding output on the console screen:

(1)



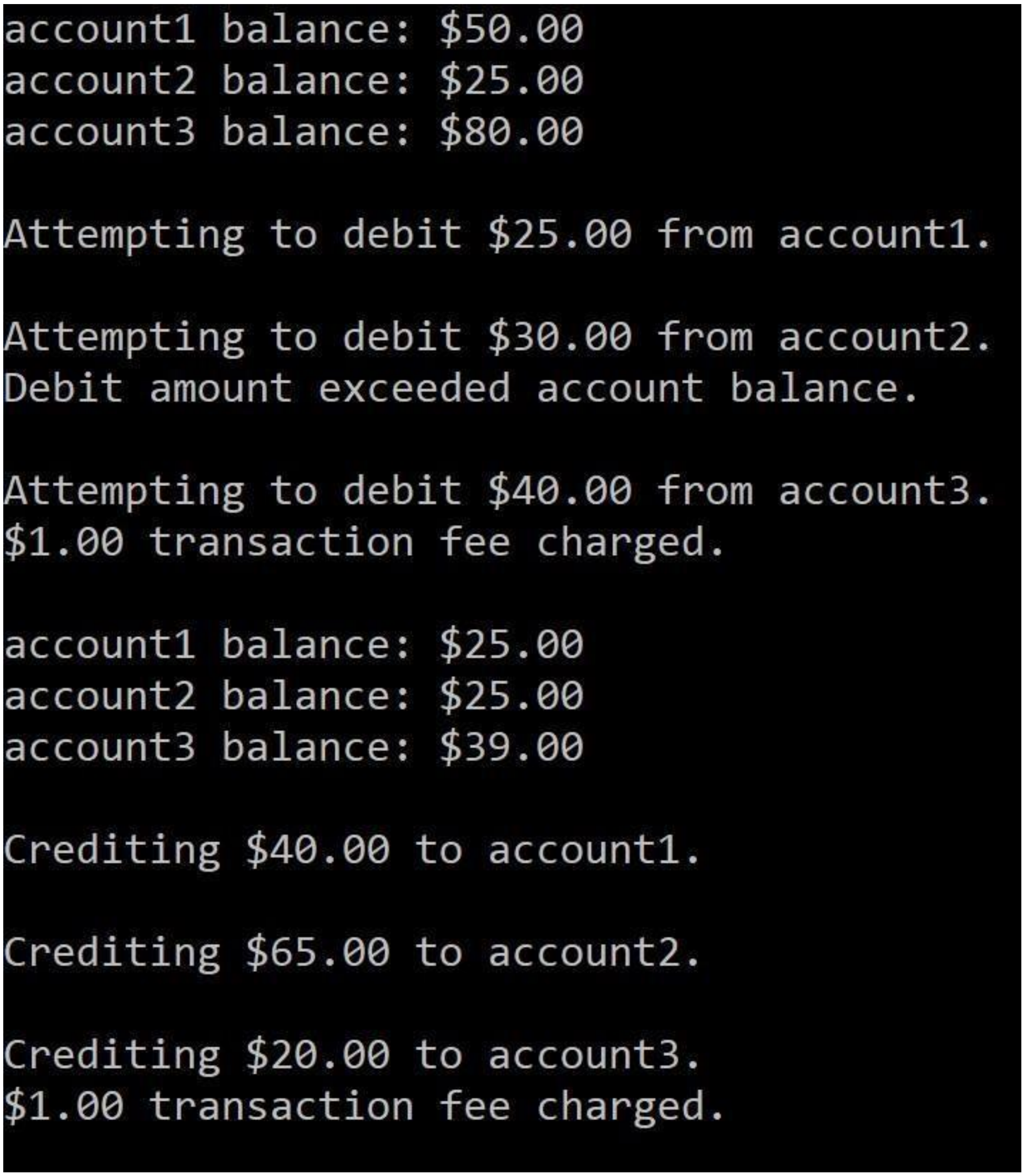
```
int main()
{
    Account account1( 50.0 ); // create Account object
    SavingsAccount account2( 25.0, .03 ); // create SavingsAccount object
    CheckingAccount account3( 80.0, 1.0 ); // create CheckingAccount object

    cout << fixed << setprecision( 2 );

    // display initial balance of each object
    cout << "account1 balance: $" << account1.getBalance() << endl;
    cout << "account2 balance: $" << account2.getBalance() << endl;
    cout << "account3 balance: $" << account3.getBalance() << endl;

    cout << "\nAttempting to debit $25.00 from account1." << endl;
    account1.debit( 25.0 ); // try to debit $25.00 from account1
    cout << "\nAttempting to debit $30.00 from account2." << endl;
    account2.debit( 30.0 ); // try to debit $30.00 from account2
    cout << "\nAttempting to debit $40.00 from account3." << endl;
    account3.debit( 40.0 ); // try to debit $40.00 from account3

    // display balances
    cout << "\naccount1 balance: $" << account1.getBalance() << endl;
    cout << "account2 balance: $" << account2.getBalance() << endl;
    cout << "account3 balance: $" << account3.getBalance() << endl;

    cout << "\nCrediting $40.00 to account1." << endl;
    account1.credit( 40.0 ); // credit $40.00 to account1
    cout << "\nCrediting $65.00 to account2." << endl;
    account2.credit( 65.0 ); // credit $65.00 to account2
    cout << "\nCrediting $20.00 to account3." << endl;
    account3.credit( 20.0 ); // credit $20.00 to account3
```

Change the values in the constructor calls

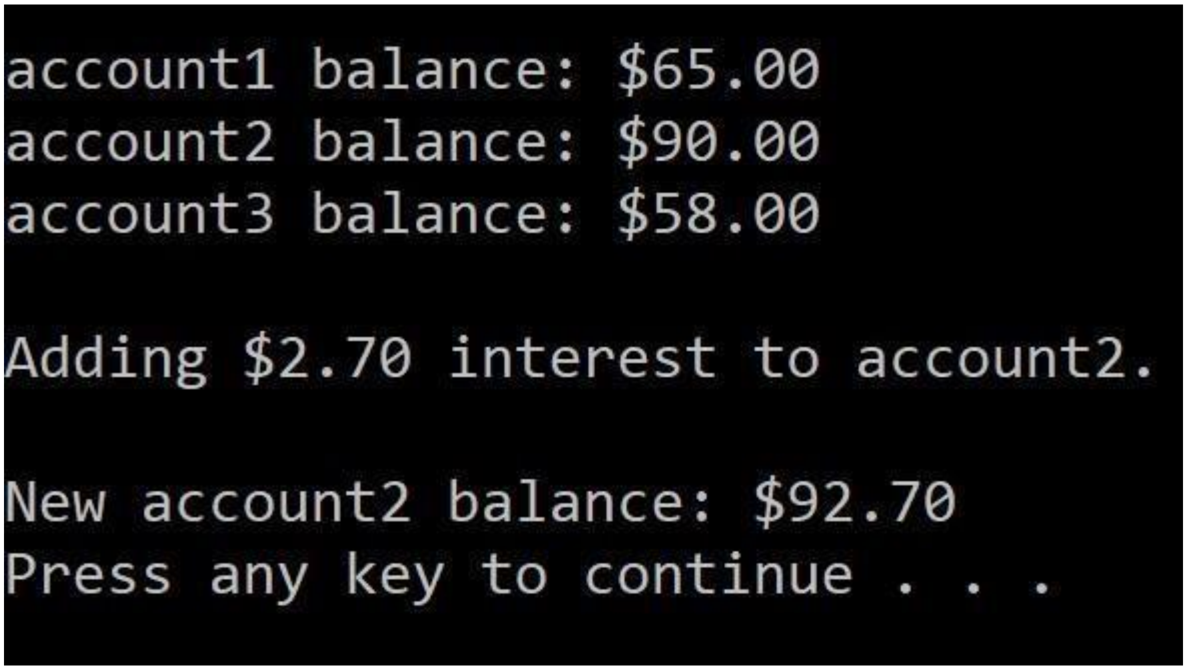
Change the values in the debit calls

Change the values in the credit calls

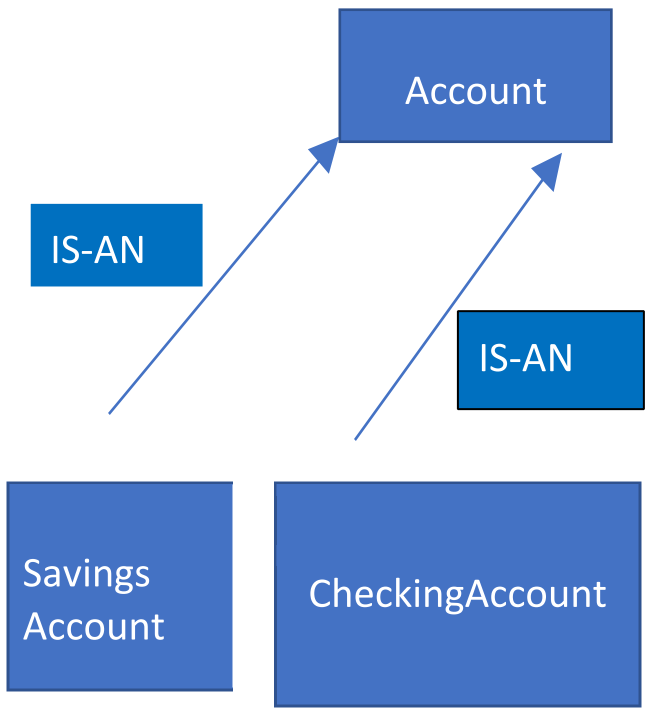
(2)



```
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 . . .
```



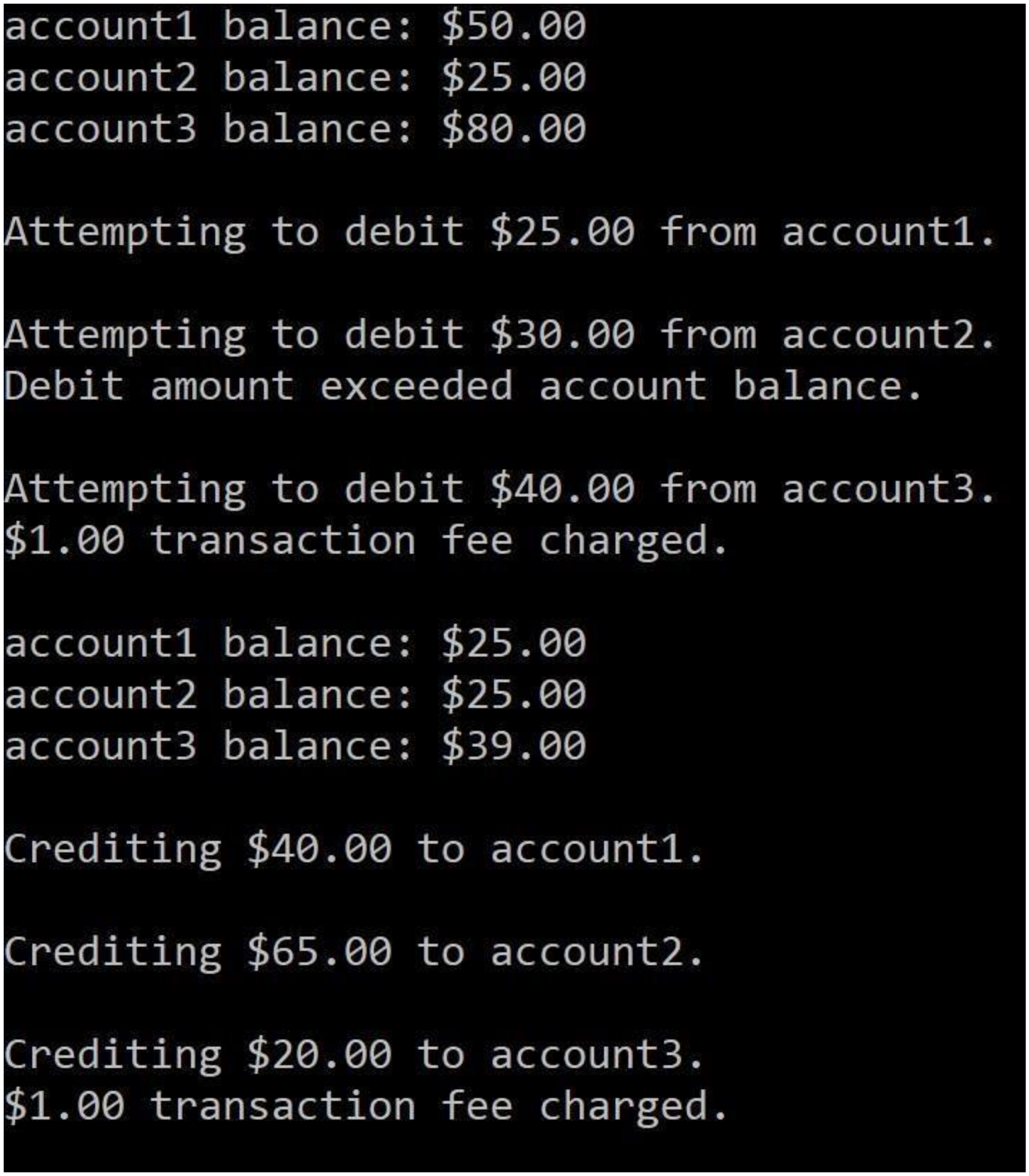
Create an **inheritance** hierarchy that a bank might use to represent customers' bank accounts. All customers at this bank can deposit (i.e., credit) money into their accounts and withdraw (i.e., debit) money from their accounts. More specific types of accounts also exist. Savings accounts, for instance, earn interest on the money they hold. Checking accounts, on the other hand, charge a fee per transaction (i.e., credit or debit).

Create an inheritance hierarchy containing base class Account and derived classes SavingsAccount and CheckingAccount that inherit from class Account. Base class Account should include one data member of type double to represent the account balance. The class should provide a constructor that receives an initial balance and uses it to initialize the data member. The constructor should validate the initial balance to ensure that it's greater than or equal to 0.0. 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. The class should provide three member functions. Member function credit should add an amount to the current balance. Member function debit should withdraw money from the Account and ensure that the debit amount does not exceed the Account's balance. If it does, the balance should be left unchanged and the function should print the message "Debit amount exceeded account balance." Member function getBalance should return the current balance.

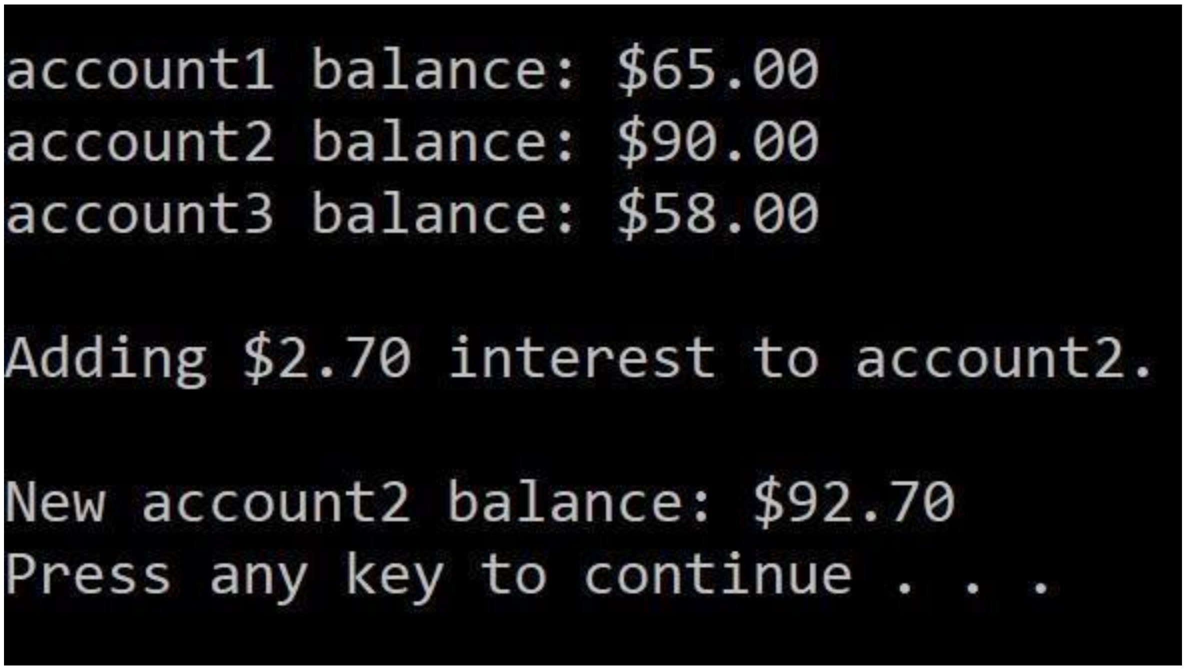
Derived class SavingsAccount should inherit the functionality of an Account, but also include a data member of type double indicating the interest rate (percentage) assigned to the Account. SavingsAccount's constructor should receive the initial balance, as well as an initial value for the SavingsAccount's interest rate. SavingsAccount should provide a public member function calculateInterest that returns a double indicating the amount of interest earned by an account. Member function calculateInterest 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.]

Derived class CheckingAccount should inherit from base class Account and include an additional data member of type double that represents the fee charged per transaction. CheckingAccount's constructor should receive the initial balance, as well as a parameter indicating a fee amount. Class CheckingAccount should redefine member functions credit and debit so that they subtract the fee from the account balance whenever either transaction is performed successfully. CheckingAccount's versions of these functions should invoke the base-class Account version to perform the updates to an account balance. CheckingAccount's debit function should charge a fee only if money is actually withdrawn (i.e., the debit amount does not exceed the account balance). *[Hint:* 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.]

Sample run (using Account_app.cpp):



```
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 . . .
```