Develop a polymorphic banking program using the Account hierarchy created in Bank Account Inheritance Hierarchy Program. First create a new account type: **OverDraftCheckingAccount**. This account type is derived from the **CheckingAccount**. Its constructor takes an initial balance, fee, overdraft fee AND a pointer to a **SavingsAccount**.

The **OverDraftCheckingAccount** allows for a **SavingsAccount** to cover debits that are too big for the OD Checking balance. If the checking account does not have enough money to cover the **withdraw()** then, for a huge fee, the linked Savings account can be used to cover the shortfall. Make sure there is enough money total to cover the **withdraw()** and both the checking fee AND the over Draft fee.

Create a vector of Account pointers to each account: two **SavingsAccount**, a **CheckingAccount** and an **OverDraftCheckingAccount** objects (which links to one of the savings accounts). Create these accounts programmatically (no user input) with various balances, transaction fees and interest rates as appropriate to that type of account. A typical Checking transaction fee is $2, where as an OverDraft fee averages around $35!
(https://www.nerdwallet.com/article/banking/overdraft-fees-what-banks-charge

Links to an external site.

)

For each account in the vector, allow the user to specify (range checked to disallow negate numbers) an amount of money to withdraw from the **BaseAccount** using member function **withdraw()**. Then allow the user to specify (range checked to disallow negate numbers) an amount of money to deposit into the Account using member function **deposit()**. As you process each Account, determine its type [Hint : Use **dynamic_cast< > w**hich implies that the base class has to be polymorphic. Which means you have defined virtual function(s)]. If the account is a **SavingsAccount**, calculate the amount of interest owed to the Account using member function **calculateInterest()**, then add the interest to the account balance using member function **deposit()**. After processing an Account, print the up-dated account balance obtained by invoking base-class member function **getBalance()**. Repeat for all accounts.

If the user enter zero for any transaction, that transaction should not cause a transaction fee to be charged.

There will be 9 files in your project: two file (.hpp and .cpp) for each account type, plus the base account. And you main.cpp file. You will submit 10 files (your output included).

Project Buddies allow. No code sharing! Just someone to ask questions / get ideas from.

Sample output:

```
Account 1 balance: $25.00
Enter an amount to withdraw from Account 1: 40
Debit amount exceeded account balance.
Enter an amount to deposit into Account 1: 10
Adding $1.05 interest to Account 1 (a Savings account)
Updated Account 1 balance: $36.05

Account 2 balance: $80.00
Enter an amount to withdraw from Account 2: 0
Enter an amount to deposit into Account 2: 20
$2.00 transaction fee charged.
Updated Account 2 balance: $98.00

Account 3 balance: $200.00
Enter an amount to withdraw from Account 3: 25
Enter an amount to deposit into Account 3: 0
Adding $2.62 interest to Account 3 (a Savings account)
Updated Account 3 balance: $177.62

Account 4 balance: $400.00
```

Enter an amount to withdraw from Account 4: 450

Debit amount exceeded account balance.

Using Overdraft Protection (Legal money extraction rip-off!).

$1.50 transaction fee charged.

$35.00 overdraft transaction fee charged.

Updated Overdraft Savings Account balance: $91.12

Enter an amount to deposit into Account 4: 25

$1.50 transaction fee charged.

Updated Account 4 balance: $23.50

Program ended with exit code: 0

Submission:

10 files

Two files (.hpp and .cpp) for each of 3 account types, plus the base account.

main.cpp

OverDraftOutput.rtf