Polymorphism and Dynamic Binding (Winter 2015)

(Account Inheritance Hierarchy) 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).

Your inheritance hierarchy must contain base class Account and derived classes SavingAccount 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 base 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 SavingAccount 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. SavingAccount's constructor should receive the initial balance, as well as an initial value for the SavingAccount's interest rate. SavingAccount 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. The amount of interest earned is then added to the SavingAccount object by passing it to the object's credit function. [*Note:* SavingAccount 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 override 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.

Develop a polymorphic banking program using the Account hierarchy created. Create a vector of Account pointers to SavingAccount and CheckingAccount objects. For each Account in the vector, allow the user to specify an amount of money to withdraw from the Account using member function debit and an amount of money to deposit into the Account using member function credit. As you process each Account, determine its type. If an Account is a SavingAccount, calculate the amount of interest owed to the Account using member function calculateInterest, and then add the interest to the account balance using member function credit. After processing an Account, print the updated account balance obtained by invoking base-class member function getBalance.

Note 1: To achieve polymorphic behavior in the Account hierarchy, each class definition must declare the debit and credit member functions as virtual functions.

Note 2: in order to determine the type of an object during the run time, you can use the C++ RTTI capabilities including, the typeid operator and type_info class. You must include header file <typeinfo> in your program to use the RTTI capabilities.