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. Check accounts on the other hand, charge a fee per transaction.

Create base class `Account` and derived classes `SavingsAccount` and `CheckingAccount` that inherit from the class `Account`. Base class `Account` should include one private instance variable of type decimal to represent the account balance. The class should provide a constructor that receives an initial balance and uses it to initialize the instance variable with a public property. The property should validate the initial balance to ensure that it's greater than or equal to 0.0; if not, throw an exception. The class should provide two public methods. Method `Credit` should add an amount to the ***current balance***. Method `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 **method** should `return` a `false`. The class should also provide a `get` accessor in property `Balance` that returns the current balance.

Derived class `SavingsAccount` should inherit the functionality of an Account, but also include a decimal instance variable 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 interest rate. `SavingsAccount` should provide public method `CalculateInterest` that returns a decimal indicating the amount of interest earned by an account. Method `CalculateInterest` should determine this amount by multiplying the interest rate by the ***account balance***. [*Note*: `SavingsAccount` should inherit methods `Credit` and `Debit` without redefining them.]

Derived class `CheckingAccount` should inherit from base class `Account` and include a decimal instance variable 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 methods, `Credit` and `Debit`, so that they subtract the fee from the account balance whenever either transaction is performed successfully. `CheckingAccount`'s versions of these methods should invoke the base-class `Account` version to perform the updates to the account balance. `CheckingAccount`'s `Debit` method 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` method so that it returns a `bool` indicating whether the money was withdrawn. Then use the return value to determine whether a fee should be charged.]

After defining the class in this hierarchy, write an app that creates objects of each class and tests their methods. Add interest to the `SavingsAccount` object by first invoking its `CalculateInterest` method, then passing the returned interest amount to the object's Credit method.

<u>**ENSURE**</u> that <u>**all**</u> ***input amounts*** and ***interest rates*** to your **methods** and **properties** are all ***non-negative values***. That is, if any of them turns out to be ***negative***, simply throw an `ArgumentOutOfRangeException` exception in the **methods** and the **properties**.