Inheritance Exercises

Bank Teller Application

This is a two day set of exercises. Parts I is to be completed on Day One. Parts II and III should be finished on Day Two.

Day One - Part I

Create three new classes to represent a bank account, savings account, and a simple checking account.

1) BankAccount

The BankAccount class represents a simple checking or savings account at a bank. The balance is represented in USD using the DollarAmount type.

1. Implement the `BankAccount` class.

Property Name	Data Type	Get	Set	Description
AccountNumber	string	X	X	Returns the account number that the account belongs to.
Balance	decimal	X	private	Returns the balance value of the bank account.

Method Name	Return Type	Description
Deposit(decimal amountToDeposit)	decimal	Adds amountToDeposit to the current balance, and returns the new balance of the bank account.
Withdraw(decimal amountToWithdraw)	decimal	Subtracts amountToWithdraw from the current balance, and returns the new balance of the bank account.
Transfer(BankAccount destinationAccount, decimal transferAmount)	void	Withdraws `transferAmount` from this account and deposits it into `destinationAccount`.

Constructor	Description
BankAccount()	A new bank account's balance is defaulted to a 0 dollar balance.

```
//Sample Usage
BankAccount b1 = new BankAccount();
BankAccount b2 = new BankAccount();

decimal amountToDeposit = 100.00M;
```

```
decimal newBalance = b2.Deposit(amountToDeposit);

decimal amountToTransfer = 50.0M;
b2.Transfer(b1, amountToTransfer);
```

2. Write unit tests to validate the state and functionality of your public methods.

2) CheckingAccount

CheckingAccount extends the BankAccount class you just created, plus the following additional rules:

1. Implement the `CheckingAccount` class.

Ousing System.Collections.Generic;	**Description**
Withdraw	If the balance falls below $0.00 a $10.00 overdraft fee is also withdrawn from the account.
Withdraw	Checking account cannot be more than $100.00 overdrawn. If a withdrawal is requested leaving the account more than $100.00, it fails and the balance remains the same.

2. Write unit tests to verify the functionality of your code.

3) SavingsAccount

SavingsAccount extends the BankAccount class you just created, plus the following additional rules:

Override Method	**Description**
Withdraw	If the current balance is less than $150.00 when a withdrawal is made, an additional $2.00 service charge is withdrawn from the account.
Withdraw	If a withdrawal is rquested for more than the current balance, the withdrawal fails and balance remains the same.

2. Write unit tests to verify the functionality of your code.

Day Two - Part II

This is the Day Two continuation of the Bank Teller Application exercise.

Create a new class that represents a bank customer.

1) BankCustomer

1. Create the BankCustomer class to represent a bank customer.

Property Name	Data Type	Get	Set	Description
Name	string	X	X	Returns the account holder name that the account belongs to.
Address	string	X	X	Returns the account number that the account belongs to.
PhoneNumber	string	X	X	Returns the account number that the account belongs to.
Accounts	BankAccount[]	X		Returns the customer's list of BankAccount objects as an array.

Method Name	Return Type	Description
AddAccount(BankAccount newAccount)	void	Adds newAccount to the customer's list of accounts.

2. Write unit tests to verify the functionality of your code.

Day Two - Part III

Customers whose combined account balances are at least $25,000 are considered VIP customers and receive special privileges.

1. Add a `bool IsVIP` property (you may use a function too if you wish) to the bank customer class that returns true if the sum of all accounts belonging to that customer is at least $25,000 and false otherwise.

Write unit tests to verify the functionality of your code.