Inheritance Pair Exercises

Bank Teller Application

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

Day One - Part 1

DollarAmount

The DollarAmount class is included in the project provided for this exercise.

The DollarAmount represents an amount of currency in US Dollars. It is meant to be used in place of primitive double or decimal point types in order to represent monetary amounts accurately.

This class is immutable, meaning that no function will change the internal state of an object.

Attribute Name	Data Type	Get	Set	Description
cents	int	X		Amount of cents after the decimal point (e.g. $1.25 is 25 cents)
dollars	int	X		Amount of whole dollars before the decimal point (e.g. $1.25 is 1 dollar)
isNegative	bool	X		If the object represents a negative value DollarAmount

Method Name	Return Type	Description

plus(DollarAmount amountToAdd)	DollarAmount	Adds the value of "this" DollarAmount object to the amountToAdd instance. The total is returned as a new DollarAmount
minus(DollarAmount amountToSubtract)	DollarAmount	Subtracts the value of amountToSubtract instance from "this" instance. The difference is returned as a new DollarAmount

Constructor	**Description**
DollarAmount(int totalCents)	Instantiates a new DollarAmount object that represents totalCents passed in.
DollarAmount(int dollars, int cents)	Instantiates a new DollarAmount object that represents dollars and cents.

Exercise

1. Override the inherited implementation of the toString() function of the DollarAmount class included in the project provided for this exercise.

> Example output of the overridden toString() function:
>
> ```
> new DollarAmount(3210).toString()  →  “$32.10”
> new DollarAmount(1000).toString() → “$10.00”
> new DollarAmount(1).toString() → “$0.01”
> ```

2. Write unit tests to verify the functionality of the class.

Day One - Part II

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.

Attribute Name	Data Type	Get	Set	Description
accountNumber	String	X	X	Returns the account number that the account belongs to.
balance	DollarAmount	X		Returns the balance value (represented as a DollarAmount object) of the bank account.

Method Name	Return Type	Description
deposit(DollarAmount amountToDeposit)	DollarAmount	Adds amountToDeposit to the current balance, and returns the new balance of the bank account.
withdraw(DollarAmount amountToWithdraw)	DollarAmount	Subtracts amountToWithdraw from the current balance, and returns the new balance of the bank account.

transfer(BankAccount destinationAccount, DollarAmount 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();
DollarAmount amountToDeposit = new DollarAmount(100);
DollarAmount newBalance =
b2.Deposit(amountToDeposit);
DollarAmount amountToTransfer = new DollarAmount(50);

b2.Transfer(b1, amountToTransfer);
```

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

2) CheckingAccount

CheckingAccount has all of the same behavior of the BankAccount class you just created, plus the following additional rules:

1. Implement the CheckingAccount class.

Override Method	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 has all of the same behavior of 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 III

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.

Attribute 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 IV

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

1. Add a boolean isVIP *derived* attribute 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.