Inheritance Individual Exercises

The purpose of this exercise is to practice writing code that uses the Object-Oriented Programming principle of inheritance.

Learning objectives

After completing this exercise, students will be able to:

- Describe the purpose and use of inheritance in an Object-Oriented Programming environment
- Define and use superclasses and subclasses in an inheritance hierarchy
- Identify superclasses and subclasses from viewing source code
- Define an IS-A relationship in reference to inheritance
- Define what overriding means in the context of inheritance
- Describe what's being inherited
- Describe how access modifiers work in an inheritance relationship
- Use super class constructors in a subclass

Evaluation criteria and functional requirements

- The project must not have any build errors.
- Code is presented in a clean, organized format.
- Code is appropriately encapsulated.
- Inheritance is used appropriately to avoid code duplication.
- The code meets the specifications defined below.

Bank teller application

Notes for All Classes

- All attributes have `private` access.
- X in the get column indicates the attribute **should have a `get` accessor**.
- X in the set column indicates the attribute **should have a `set` accessor**.

Instructions

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

1. Implement the `BankAccount` class

The `BankAccount` class represents a simple checking or savings account at a bank.

Constructor	Description
BankAccount(String accountHolderName, String accountNumber)	A new bank account requires an account holder name and account number. The balance defaults to a 0 dollar balance if not specified.

Constructor	Description
BankAccount(String accountHolderName, String accountNumber, int balance)	A new bank account requires an account holder name and account number. The balance is initialized to the dollar balance given.

Attribute Name	Data Type	Get	Set	Description
accountHolderName	String	X		Returns the account holder name that the account belongs to.
accountNumber	String	X		Returns the account number that the account belongs to.
balance	int	X		Returns the balance value of the bank account in dollars.

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

2. Implement the `CheckingAccount` class

A `CheckingAccount` "is-a" `BankAccount`, but it also has some additional rules:

Override Method	Description
withdraw	If the balance falls below $0, a $10 overdraft fee is also withdrawn from the account.
withdraw	Checking account can't be overdrawn by $100.00 or more. If a withdrawal request leaves the account $100 or more overdrawn, it fails and the balance remains the same.

3. Implement the `SavingsAccount` class

A `SavingsAccount` "is-a" `BankAccount`, but it also has some additional rules:

Override Method	Description
withdraw	If the current balance is less than $150 when a withdrawal is made, an additional $2 service charge is withdrawn from the account.
withdraw	If a withdrawal is requested for more than the current balance, the withdrawal fails and balance remains the same. No fees are incurred.

Sample usage

```
BankAccount checkingAccount = new CheckingAccount("Bernice", "CHK:1234");
BankAccount savingsAccount = new SavingsAccount("Bernice", "SAV:9876");

int amountToDeposit = 2;
int newBalance = checkingAccount.deposit(amountToDeposit);
```

Challenge

The industry standard way to deal with decimal numbers in Java is with the `BigDecimal` class. Can you change the `balance` variables in the classes above to use `BigDecimal` instead of `int`?

Tips and tricks

- A good way to determine if you're implementing inheritance correctly is to read the code or classes out loud. A child class "is-a" type of its parent. For instance, a CheckingAccount "is-a" BankAccount. Is a BankCustomer a BankAccount, or does a BankCustomer have a BankAccount? Thinking about the relationships of objects in these terms helps you to quickly identify opportunities to improve your code.

- derived-properties
- inheritance-and-an-is-a-relationship
- oop-inheritance
- what-is-polymorphism