Lab 4: Bank Account Polymorphism

Due:

See the due date on the Blackboard of your Lab.

Purpose

This is an exercise to practice with the concept of Polymorphism. Suggested Reading: Chapters 9, 10 of Deitel and Deitel.

Description

This is a polymorphism example, so right from the start we know we will be processing items of a type that is a super-class (maybe abstract) or an interface.

In this case for this exercise we have bank accounts. What methods would bank accounts have? **calculateInterest**? **deposit**? **withdraw**? **getBalance**? **calculateAndUpdateBalance**? What kinds of bank accounts are there? **Chequing**, **Savings**, **Investment**, **Credit Card**? As programmers by now we have been trained to recognize that there can be some methods that would be done the same way in all cases, implemented with the same Java code (**getBalance**?) and there can be some methods that might be done differently depending on what sub-type it is, what type of account it is (**calculateAndUpdateBalance**?) Similarly, the attributes (also known as instance variables) can be divided up into two kinds: those that are the same no matter what type of account it is, and those that depend on the subclass or type of account. The actual balance could be represented by a double in all cases, but the fees might not even exist for some bank account types, so fees would be present as an attribute in only some of the subclasses, perhaps. (I'm saying "perhaps" and "maybe" because we aren't bankers -- if we were programming for a bank, their banking experts would spell out all of these details, and our job as programmers is to know how best to represent those details when they're spelled out to us.) The basic question is what's the same for all of them, and what is not the same depending on the specific type.

So, based on the above, we have the following summary to use for the present exercise:

(super, abstract) class **BankAccount implements edu.ac.banklib.BankInterface**
attributes: **balance**
methods: **getBalance, deposit**
abstract methods: **calculateAndUpdateBalance**

class **SavingsAccount** extends **BankAccount**
attributes: **interestRate** (yearly)
overrides **calculateAndUpdateBalance** (add the interest for the month)

class **ChequingAccount** extends **BankAccount**
attributes: **fee** (monthly)
overrides **calculateAndUpdateBalance** (subtract the fee)

We can now think about the polymorphic processing. Suppose we have thousands of accounts of various types and we need to calculate the updated balance every month. To keep things simple, we'll just add interest earned by a savings account, or charge the monthly fee for a chequing account. Every month we would run a loop that goes through a collection of accounts and invokes the **calculateAndUpdateBalance** method on each of these accounts. It is polymorphic processing because the type of the objects we are processing is a superclass or an interface (in this case, a superclass). The details of the **calculateAndUpdateBalance** method come from the sub-class.

Steps

1. Implement BankAccount, SavingsAccount and ChequingAccount as Java classes, **including javadoc comments**. The fee amount and interestRate, must be obtained from the **InitialValues** class in BankLib library (download from Blackboard)
2. Create a **BankAccountTest** class that declares, instantiates, and initializes a single array of 30 bank accounts (first 15: chequing account, and the rest savings account). Each bank account has an initial value obtained from **InitialValues** array of initialAccount (again from BankLib library). You need to use the deposit method to initialize the balance.
    a. Add a method **monthlyProcess** that takes an array of bank accounts as a parameter, and does the monthly balance update for each account.
    b. Add a method **printBalance** that uses **BankUtil**.printBalance method in BankLib library to print out balance in neat common format. See printBalanceJavadoc.pdf (from BlackBoard).
    c. Add a main method that creates an instance of the **BankAccountTest** class, and calls the **monthlyProcess** and **printBalance** methods 12 times (simulating an entire year)

Demonstration and Submission

Demonstrate and submit your program and Javadoc folder as per usual zip file convention, before the due date on Blackboard.

Demonstration: 4 marks (both the program output, and Javadoc)
Javadoc: 3marks
Java code: 3 marks