CEJV416 OOP with Java **Lab#2**

The purpose of this problem is to develop a hierarchy of classes that represent different types of bank accounts and then to simulate the most common transactions upon these accounts.

The first step is to create a data fields that holds the following basic information about a bank account:

`Starting balance this month`
`Current balance this month`
`Total of deposits this month`
`Number of deposits this month`
`Total of withdrawals this month`
`Number of withdrawals this month`
`Annual interest rate`
`This month's service charge`
`Current account status` (to represent an active or inactive account)

Create an abstract class that defines the basic operations of the banking system. It will have as a class variable all the data fields. Make the data fields protected.

The abstract super class should have the following member methods:

`Constructor`: Accepts arguments for the balance and annual interest rate

`makeDeposit`: A method that accepts an argument for the amount of the deposit. The method should add the argument to the account balance. It should also increment the variable holding the number of deposits.

`makeWithdraw`: A method that accepts an argument for the amount of the withdrawal. The method should subtract the argument from the balance. It should also increment the variable holding the number of withdrawals.

`calculateInterest`: A method that updates the balance by calculating the monthly interest earned by the account, and adding the interest to the balance. This is performed by the following formulas:

```
Monthly Interest Rate = (Annual Interest Rate / 12)
Monthly Interest = Balance * Monthly Interest Rate
Balance = Balance + Monthly Interest
```

`doMonthlyReport`: A method that subtracts the monthly service charges from the balance, calls the calculateInterest method, and then sets the variables that hold the number of withdrawals, number of deposits, and monthly service charges to zero.

Create a savings account class that is a subclass of the abstract account class. It should have the following member methods:

`makeWithdraw`: A method that checks to see if the account is inactive before a withdrawal is made. A withdrawal is then made by calling the super class version of the method, if permitted.

`makeDeposit`: A method that checks to see if the account is inactive before a deposit is made. If the account is inactive and the deposit brings the balance above $25, the account becomes active again. The deposit is made by calling the super class version of the method.

`doMonthlyReport`: Before the super class method is called, this method checks the number of withdrawals. If the number of withdrawals for the month is more than 4, a service charge of $1 for each withdrawal above 4 is added to the monthly service charge in the bean.

If the balance of a savings account falls below $25, it becomes inactive (status is false) . No more withdrawals may be made until the balance is raised above $25, at which time the account becomes active again.

Create a checking account class that is a subclass of the abstract account class. It should have the following member methods:

`makeWithdraw`: Before the super class method is called, this method will determine if a withdrawal (a check written) will cause the balance to go below $0. If the balance goes below $0, a service charge of $15 will be taken from the account. The withdrawal will not be made due to insufficient funds. If there isn't enough in the account to pay the service charge, the balance will become negative and the customer will owe the negative amount to the bank.

`doMonthlyReport`: Before the super class method is called, this method adds the monthly fee of $5 plus $0.10 per withdrawal to the monthly service charge in the bean.

The program will allow the end user to select an account type and make deposits and withdrawals. The reports show the activities for the month for the chosen account type. This is the starting balance, total amount of deposits, total amount of withdrawals, service charges, current balance and account status. After a report is displayed transfer the current balance to the starting balance and zero out all the other variables except starting balance and interest. Use the following menus:

```
Bank Menu         Savings Menu            Checking Menu
A: Savings        A: Deposit              A: Deposit
B: Checking       B: Withdrawal           B: Withdrawal
C: Exit           C: Report               C: Report
                  D: Return to Bank Menu  D: Return to Bank Menu
```

You may add any additional classes, methods and variables as required.