CET1012 - Programming Methodologies: Java - Practicum 03

Topics Covered: Classes and Objects

Learning Objectives:

- Familiarize with basic Java classes and objects. This includes
  - constructors
  - methods
  - proper usage of access & non-access modifiers
- Apply coding best practices such as
  - proper documentation using Javadoc

must find out how to do Javadoc.

Deliverables:

- Submit a single Java file called `DigiBankAccount.java`
- Include your name in your `.java` file at the top.
- Note that a non-working submission will result in a zero

Background

A bank account is a financial account maintained by a bank in which a financial transaction between the bank and a customer are recorded. Examples of financial transaction includes deposits and withdrawals. The typical bank account also provides a convenient way of managing your finances by keeping a track of transaction history.

Task

Your task is to implement a simple banking account class named `DigiBankAccount`

Your class should have the following data fields:

1. `accountName`
   - This variable holds the value of the account holder
2. `accountNumber`
   - This variable holds the value of the account number
   - The 8-digit account number is automatically generated when a new account is made based on the following formula:

     $\text{new account number} = 1234 + \text{number of accounts existing}$

     Example:

     the first account will hold the following account number: $1234\ 0001$, followed by $1234\ 0002$ and so on.
3. `balance`
   - This variable holds the total amount of money in the account.
   - You should also note that you should not use a floating point representation for money
4. `transactionHistory`
   - This variable that holds up to 5 transactions (deposit/withdrawal). If no transaction, have been done, the default value should be `null`
5. `numberOfAccounts`
   - This holds the number of accounts in total
6. any other data fields that you may required

Your class should have the following methods:

1. `DigiBankAccount`
    - constructor invoked when a new account is created
    - takes in the name of the account holder as input
    - assigns an initial value to data fields such as `accountNumber` and `balance`
2. `deposit`
    - takes in any valid monetary amount in dollars as input e.g. $9.99 or $9.998. (note: you may exclude the '$' symbol & $9.998 is a valid input as only the first 2 decimal places will be used and the rest truncated).
    - updates `transactionHistory` on every successful deposit (up to 5 deposit/withdrawals)
3. `withdraw`
    - takes in any valid monetary amount in dollars as input e.g. $9.99 or $9.998. (note: you may exclude the '$' symbol & $9.998 is a valid input as only the first 2 decimal places will be used and the rest truncated).
    - updates `transactionHistory` on every successful withdrawal (up to 5 deposit/withdrawals)

you need to only show 5 deposits/withdrawals if got 6 then just overwrite the last one.

4. `getNumberOfAccounts`
    - getter method that returns `numberOfAccounts`
5. `displayBalance`
    - a method that displays the `balance`
6. `displayTransactionHistory`
    - a method that displays the `transactionHistory`
    - do not update `transactionHistory` should any transaction fail
7. any other methods that you may require

Program Requirements

- You may assume that the inputs entered are valid within the numerical number range.
- An account may not have negative balance
- You may assume that the account will only make a maximum of 5 valid transactions.
- Your program will display the `balance` and `transactionHistory` in dollars (2 decimal places)

Below is a sample output:

Assuming 2 accounts have been created with the names being `a` and `b` respectively, here are some of the expected outputs after calling `a.displayTransactionHistory()` and `b.displayTransactionHistory()`:

```
Account Name: a
Account Number: 12340001
Balance: 300.00
1. deposit 100.00
2. withdraw 100.00
3. deposit 100.00
4. deposit 100.00
5. deposit 100.00

Account Name: b
Account Number: 12340002
Balance: 0.00
1. deposit 200.00
2. deposit 200.00
3. withdraw 200.01
4. deposit 200.00
5. withdraw 399.99
```

For the maximum allocation of marks, refer to the table below.

Description	**Marks (%)**
Successful implementation of the Class	45
Proper usage of access and non-access modifiers	12
Program able to handle edge cases	25
Proper and sufficient comments to explain code using Javadoc	8
Proper and consistent naming conventions.	5
Proper display outputs (easy to read, correct decimal places, etc.)	5

Once you have completed, save your file in the following format `DigiBankAccount.java`. Include your name at the top of the file.

just create the DigiBankAccount.java then test it using Main.java but only submit the DigiBankAccount.java