Lab 10

Requirements:

- Create a Java project named yourStudentId_OOP_Lab10
- Read instructions and create classes needed. You are supposed to add 4 classes (*BankAccount, CheckingAccount, SavingsAccount,* and *Tester*) to the project.
- All instance variables are private. Please use public interfaces to access private variables.

Description:

Nowadays, people may not have just one bank account but have different types of the bank accounts. A savings account can be used to deposit money with extra interests. The interest will be added into the balance at the end of each month. A checking account is for the use of everyday transactions such as purchases, bill payments and ATM withdrawals. Each month a user can have three free transaction number from the checking account. If a user had withdrawn more than three times from the checking account, there is a commission charge for each extra withdraw. These two types of accounts have similar attributes inherited from a general bank account, but also have some different attributes and methods for themselves. Please check the following class diagram and implement them.

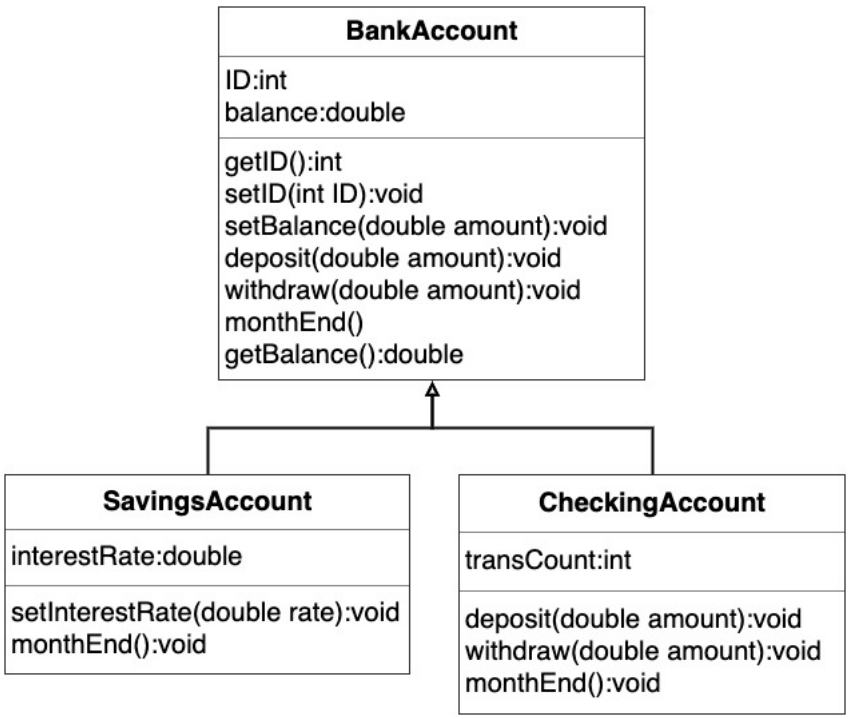

Figure 1.

1. **Create *BankAccount* class**

BankAccount	
Modifier and type	**Method (or Variable) and description**
Instance variable	
int	ID The account's ID.
double	balance The account's existing balance
Instance methods	
-	1 getter for 1 attributes (getID()). 2 setters for 2 attributes (setID(...), setBalance(...)).
void	deposit(double amount) a. Add amount to balance and update.
void	withdraw(double amount) a. Determine whether the user's balance is greater than the amount will be withdrawn. If it is *"true"*, you should subtract the amount from the balance and update it. Otherwise, print out *"Your account does not have enough money."*.
void	monthEnd()
double	getBalance() Return the balance of the user's account.

2. **Create *SavingsAccount* class**

SavingsAccount	
Modifier and type	**Method (or Variable) and description**
Instance variable	
double	interestRate The monthly interest for the saving account.
Constructor	
SavingsAccount(int amount, int ID, double interestRate) Enable to instantiate the object of *SavingAccount* with given *amount*, *ID*, and *interestRate*.	
Instance methods	
-	1 setter for 1 attributes (setInterestRate(...)).
void	monthEnd() a. Update the total balance with monthly interest rate. b. The monthly interest rate is expressed in percentage. For example, the monthly interest rate "1%" will be regarded as 0.01.

3. **Create *CheckingAccount* class**

CheckingAccount	
Modifier and type	**Method (or Variable) and description**
Instance variable	
int	transCount The number of monthly transaction for the account.
Constructor	
CheckingAccount(int amount, int ID) Enable to instantiate the object of *CheckingAccount* with given *amount* and *ID.*	
Instance methods	
void	deposit(double amount) a. Update the number of transaction. b. Update the balance of the account.
void	withdraw(double amount) a. Update the number of transaction. b. Update the balance of the account.
void	monthEnd() a. Initialize the double variable named *commissionFee* as 5. b. Determine whether the number of transactions is greater than three times, and if so, deduct the commision fee from the balance. c. Reset the number of *transCount* as 0.

4. **The *main* method in *Tester* class**

   a. Create four bank account objects. Two of them belong to *SavingsAccount* class named *accountA* and *accountB*, and the others belongs to *CheckingAccount* class named *accountC* and *accountD*. The *amount*, *id*, and *interestRate* as below:

      - *accountA*: (8000, 306049001, 1)
      - *accountB*: (1000, 306049011, 2)
      - *accountC*: (9000, 306016033)
      - *accountD:* (3000, 306016041)

   b. Create an array named *accounts* to store all the accounts mentioned above.

   c. Use *Scanner* methods and *"if...else"* statement to deal with the four type of action. If the input value equals to "D" will take the deposit action, "W" will take the withdrawal action. Moreover, it needs to determine which account and what amount the user will operate. The account needs to be stored in the *accounts* array. Finally, update and print out the existing balance. If user's account doesn't be included, it should print out *"Corresponding account cannot be found."* on the console.

   d. If the input value equals to "M", you need to simulate the behavior that will occur at the end of the month. You can use *"for-loop"* concept to call the corresponding method.

   e. The program still execute until the input equals to "Q".

Sample output

```
D)Deposit  W)Withdraw  M)Month end  Q)Quit:D
Enter your account number:306049001
Enter transaction amount:10000
Balance: 18000.0
D)Deposit  W)Withdraw  M)Month end  Q)Quit:W
Enter your account number:306049001
Enter transaction amount:5000
Balance: 13000.0
D)Deposit  W)Withdraw  M)Month end  Q)Quit:M
0 13130.0
1 1020.0
2 9000.0
3 3000.0
D)Deposit  W)Withdraw  M)Month end  Q)Quit:Q
```

Submission: Submit your project as ".zip file" via Moodle. No other submissions will be graded.
Reminder: Please zip the whole project
Deadline: Tomorrow's midnight (for both Mon56 and Tue23)