Object-Oriented Programming Lab#9, Fall 2021

Today's Topics

- Inheritance
- encapsulation
- method override
- method overload
- subclass polymorphism
- abstract class
- Adding project reference

A Banking System

Create a **Banking System**, where a user can **create new account**, **deposit** money, **withdraw** money, **check** the balance, and **view** the transaction records. The application will be used by both the **bank employee** and **customer/account holder**.

There are different types of **BankAccount** a user can create. See below for the requirements of different types of account.

- **Savings account:** A savings account allows user to accumulate ***interest*** on funds he has saved for future needs. Savings account required a ***minimum balance***. For our purpose let's assume the **minimum balance** is 2000 Tk and **interest rate** is 5%. From savings account, user is only **allowed to withdraw a maximum amount** of money which will be set up during the account creation.
- **Student account:** This is a **Savings** account where the ***minimum balance*** is 100 Tk and students can only user card to cash money. So, ***maximum withdraw*** limit is the limit of ATM card which is 20,000 TK.
- **Current account:** Current account offers easy access to your money for your daily transactional needs and help keep your cash secure. You need a **trading license** to open a Current account. There is no restriction on how much money you can withdraw from Current account but you need a ***minimum balance*** of 5000 TK in your account.

The system will have the following functionalities.

1. There are 2 types of users for this system; **a bank employee and account holder**. A user can log in as a bank employee or account holder. For simplicity, we can **skip** the log-in part and add an option or button for logging in as an employee or account holder. An account holder/customer will have some read-only functionalities in the system where as the Bank Employee will be able to access most of functionalities.

2. Bank Employee will have the following functionalities
    a. Add new BankAccount into in the system.
    b. Deposit money to an account on behalf of the customer.
    c. Withdraw money from a specific account on behalf of customer.
    d. Transfer money from one account to another
    e. View the summary of an account (name, nid, account number, type, and balance)
    f. View the transaction details of a specific account.
    g. View the list of specific type of Bank Accounts (Savings or Current or Student)
    h. View the list of all accounts.

3. The customer will have the following functionalities.
    a. View the summary of one of his/her account.
    b. View the transaction details of one of his/her account.
    c. View the list of his/her accounts.

*What you need to do: (**Note: Do not use default package**)*

You need 2 projects for this Lab.

Create a Project name BankingLibrary (and do the following)

1. Create a class name **Transaction**:
    a. Add 3 private instance variables; ***transactionTime(***(java.time.LocalDateTime type), ***amount***, *and* ***transactionType***
    b. Implement a parameterized constructor. - pass all attributes except ***transactionTime*** and set ***transactionTime*** to LocalDateTime.Now;
    c. Override **toString()** method and return the string in the format "***transactionTime***\t***amount***\t\t***transactionType"*** [here the String format of ***transactionTime*** should be passed. Code for **LocalDateTime** to **String** conversion is given below]

Code to convert time to string [Use this in toString() method]

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
String appTime = transactionTime.format(formatter);
```

Note: `DateTimeFormatter` is under `java.time.format` package.

2. Create an abstract **BankAccount** class:
    a. Add 6 private instance variables; ***memberName***, ***memberNID***, ***accountNumber***, ***accountBalance***, ***minimumBalance,*** *and* ***transactions*** *(an ArrayList of type Transaction to store the transactions).*
    b. Implement constructor. You need to pass ***memberName***, ***memberNID***, ***accountBalance*** & ***minimumBalance*** as parameter.
        - *You need to auto-generate a 5 digit* ***accountNumber*** *inside the constructor. So, you do not need to pass the* ***accountNumber*** *as a parameter in the constructor.* (See the example below for how to generate 5 digit random number)

    Add the following methods inside the class:

    a. **public void deposit(double depAmount)**

        - Call ***deposit(double, String)*** and pass **depAmount** as **depAmount** and "Deposit" as the ***transactionType***.

b. **protected void deposit(double depAmount, String transactionType)**

- Inside the method the ***accountBalance*** need to be increased by the "***depAmount***" amount.
- Call **addTransaction(…)** and pass the ***depAmount*** as the amount and **transactionType** as the **transactionType**.

c. **public void withdraw(double withAmount)**

- Call ***withdraw(double, String)*** and pass **withAmount** as **withAmount** and "Withdraw" as the ***transactionType***.

d. **protected void withdraw(double withAmount, String transactionType)**

- The ***accountBalance*** is decreased by "***withAmount***" amount. We have to make sure the balance does not become less than ***minimumBalance***.
- Call **addTransaction(…)** and pass ***withAmount*** as the amount and **transactionType** as the type.

e. Add **public getter** method for ***accountNumber***, ***accountBalance*** attributes and getter/setter method for other attributes.

f. **public void addTransaction(double amt, String type)**

- Create an object of ***Transaction*** class with ***amt***, and ***type*** as the transaction parameters and add the object to ***transactions*** list.

g. **Override toString()** method

- From the method, return a String in the format "accountType: memberName\tmemberNid\taccountNumber\taccountBalance\tminimumBalance" where **accountType** is the class name (SavingAccount/CurrentAccount/StudentAccount)
  Note: use ***getClass().getSimpleName()*** to get the class name.

h. **public ArrayList<Transaction> getTansactions()**

- This method will return ***transactions*** variable.

Code to generate 5 digit random number: (3 different examples below)

*The **num** variable in the examples below will store a 5 digit number in String format.*

***Example1**:*

```
Random rand = new Random();
String num ="" + rand.nextInt(10) + rand.nextInt(10)+ rand.nextInt(10)+
rand.nextInt(10)+ rand.nextInt(10);
```

***Example2**:*

```
Random rand = new Random();
String num = 10000 + rand.nextInt(89999) + "";
```

***Example3**:*

```
String num = 10000 + (int)(Math.random()*89999) + "";
```

3. Create a **SavingsAccount** class:
   a. Make this class a **subclass** of **BankAccount** class.
   b. Add **two additional private** instance variables.
      - One is "***interest***", initialize it to 5% [0.05].
      - Another variable for ***maximum withdraw*** amount limit, name it as ***maxWithLimit***.
   c. Implement constructor.

      You need to pass ***memberName***, ***accountBalance*** , and ***maxWithLimit*** as parameter. Inside the constructor, call parent class's constructor. Note: You need to make sure ***minimumBalance*** is set to 2000.
   d. Implement another **protected** constructor and pass parameter for **all attributes** except **interest** and initialize the attributes via parent constructor and normal assignment. You **do not** need to set the ***minimumBalance*** to 2000 here.

   e. Add a **private** method ***double calculateInterest()***

      Inside the method calculate the total interest (accountBalance*interest) and return the total interest.
   f. Add ***public double getNetBalance***() method.

      This method will calculate the total interest by calling ***calculateInterest()*** method and return (*accountBalance* + total interest) but it won't change the ***accountBalance*** value.
   g. Override ***withdraw(double amount)*** method.

This method will allow to withdraw money if the withdraw amount is less than the maximum withdraw limit and doesn't set the ***accountBalance*** less than ***minimumBalance*** after withdraw. So, you need to call the parent class's ***withdraw(double)*** method.

h. Override ***withdraw(double amount, String transactionType)*** method.
Call parent class's ***withdraw(double amount, String transactionType)*** method if amount is less maximum withdraw limit.

i. Override the ***toString()***

- Call the ***toString()*** method of parent class and then concatenate "\tmaxWithLimit".

j. Add getter/setter method for the additional attributes.

4. Create a **StudentAccount** class:
    a. Should **extend** the **SavingAccount** class
    b. Add 2 private instance variables; ***studentId*** and ***institutionName***
    c. Implement constructor.
        - Use the 2nd constructor of parent class and pass appropriate value for ***minimumBalance*** and ***maxWithLimit***.

5. Create a **CurrentAccount** class:
    a. Make this class as the subclass of the **BankAccount** class
    b. Add an instance variable ***tradeLicenseNumber***.
    c. Implement constructor. Note: You need to make sure ***minimumBalance*** is set to 5000.

6. Now create a class name "**Bank**" which will mimic a real Bank that holds a list of **BankAccount**. You can use an Array or **ArrayList** to hold the list of **BankAccount**. So, the class will have 2 private attributes; ***bankName*** and ***ArrayList<BankAccount> accounts***. Add the following methods to the class.
    a. Add a constructor and pass bank name as parameter. Initialize ***bankName*** inside the constructor.
    b. Add getter method for both attributes.

    Add the following methods.

    c. ***private void addAccount(BankAccount acc)***
    - Inside the method, add the ***acc*** object to the ***accounts*** list.

    d. ***public void addAccount(String memberName, String memberNid, double accountBalance,double maxWithdrawLimit)***

- Inside the method, create a ***SavingAccount*** object using the parameter provided and add the account to the list using ***addAccount(BankAccount)*** method.

e. ***public void addAccount(String memberName, String memberNid, double accountBalance, String tradeLicenseNumber)***

- Inside the method, create a ***CurrentAccount*** object using the parameter provided and add the account to the list using ***addAccount(BankAccount)*** method.

f. ***public void addAccount(String memberName, String memberNid, double accountBalance, String tradeLicenseNumber)***

- Inside the method, create a ***StudentAccount*** object using the parameter provided and add the account to the list using ***addAccount(BankAccount)*** method.

g. ***public BankAccount findAccount(String accountNum)***

- This method will loop through the list of the **BankAccount** (***accounts***) and find the account that has matching ***accountNumber*** as the parameter. If the matching **BankAccount** is available return the object otherwise return null.

h. ***public void deposit(String accountNum, double amt)***

- Inside the method call ***findAccount(String)*** to find the account with matching ***accountNum*** and then call ***deposit(double)*** method of that object.

i. ***public*** **void withdraw(String *accountNum*, double amt)**

- Inside the method call ***findAccount(String)*** to find the account with matching ***accountNum*** and then call ***withdraw(double)*** method of that object.

j. ***public*** **void transfer(String fromAccNum, String toAccNum, double amt)**

- This method will transfer money from one account to another. So, inside the method, call ***findAccount(..)*** for both ***fromAccNum*** and ***toAccNum***. If both accounts are available in the list, call ***withdraw(double,String)*** for the ***fromAccNum*** with **transactionType**="Transferred to ***toAccNum***" and ***deposit(double,String)*** for the ***toAccNum*** with **transactionType**="Received from ***fromAccNum***"

k. ***public*** **double getBalance(String *accountNum*)**

- Inside the method call ***findAccount(String)*** to find the account with matching ***accountNum***. If the account is a **CurrentAccount**, call ***getBalance()*** method; otherwise call ***getNetBalance()*** method using the object.

l. **public ArrayList<BankAccount> getAccounts()**

- return the ***accounts*** attribute.

m. **public ArrayList<BankAccount> getAccounts(String type)**

- Depending on the type return all SavingAccount or CurrentAccount or StudentAccount available in ***accounts*** attribute.

n. ***public*** **ArrayList<Transaction> getAccTransactions(String *accountNum*)**

- Inside the method call ***findAccount(String)*** to find the account with matching ***accountNum*** and then call **getTansactions()** method of that object and return the value.
-
-

Add the following account holder/customer specific methods.

o. ***public ArrayList<BankAccount> findAccounts(String memberNID)***

- This method will loop through the list of the **BankAccount** (***accounts***) and find all the accounts that have matching ***memberNID*** as the parameter. If the matching **BankAccount(s)** are available return all those accounts as an ArrayList.

p. **public BankAccount findAccount(String memberNid, String accountNumber)**

- This method will return the **BankAccount** available in ***accounts*** list with matching ***memberNid*** and ***accountNumber***. You can also use ***findAccount(String accountNumber)*** here.

q. **ArrayList<Transaction> getAccTransactions(*String memberNid,* String *accountNum*)**

- Inside the method call ***findAccount(String, String)*** to find the account with matching ***accountNum.*** If the account's ***memberNID*** is same as the ***memberNID*** parameter then call **getTansactions()** method of that object and return the value.

<u>Now create a new project BankingApp (and do the following)</u>.

1. Add project reference of the previous project.
2. Create an **application class** (that has the main method) named "**BankApp**" which will have the **main** method.
    - In the main method, ask if the user is an employee or a customer/account holder.
    - If user is an **employee**, take his/her employeeId and display the following menu to user and take necessary action.

        - Input '1' to **add** a new Account.

          You need to provide use a submenu to create different types of account. So, you have to ask for **user name**, **what type of account** he wants to open and what would be the **initial balance**. The system will create the account (***SavingsAccount*** or ***CurrentAccount*** or ***StudentAccount*** object) with a randomly generated 5 digit account number.

        - Input '2' to **deposit** to an existing account

        - Input '3' to **withdraw** from an account.

- Input ‘4’ to **transfer** money from one account to another account.
- Input ‘5’ to **display** the summary of **a specific account**. Summary will include the accountType, membeName, memberNid, accountNumber, accountBalance, and minimumBalance.
- Input ‘6’ to display the **transactions** of a specific account.
- Input ‘7’ to display the list of account of **specific type**.
- Input ‘8’ to display the list of **all** accounts.
- Input ‘0’ to **exit** the system.

- If user is an account holder/customer, ask him/her to enter the NID. Display the following menu to user and take necessary action.
  - Input ‘5’ to display the summary **of one of his/her account**. Summary will include the accountType, membeName, memberNid, accountNumber, accountBalance, and minimumBalance.
  - Input ‘6’ to display the **transactions** of one of his/her account.
  - Input ‘8’ to display the list of **all** of his/her accounts.
  - Input ‘0’ to exit the system.