Project 1: My Banking System

Objectives

This project aims to:

- Develop programming skills when creating software applications,
- Interpret UML Class diagrams and implement it into relevant code,
- Establish parent-child relationships on classes,
- Utilize interfaces in establishing class behavior, and
- Integrate all methods from the UML Class diagrams and appropriately use them in the software.

Overview

A banking system is a good example of how complex software applications are in the real world. When it comes to banks, each bank always has several accounts. However, including all account types and their functionalities are extremely high-level. As such, in this project, you are only limited to two types: Savings, and Credit Accounts.

In this project, you are only given a limited amount of code from the ***Main*** package, and all interfaces in the ***Accounts*** package. The rest of the code will be implemented by you, the developers, by referencing the given UML Class Diagram for the Banking System.

Note that all methods, attributes, classes, and interfaces must be utilized for the Banking System. Failure to comply will result in difficulties in completing the project, and also point deductions.

Project File Structure

Firstly, download the `Project1.zip` file and extract the contents. Once this is done, it should have the following file structure:

```
|- Project 1
|--- Accounts
|----- Deposit.java
|----- FundTransfer.java
|----- IllegalAccountType.java
|----- Payment.java
|----- Transaction.java
|----- Withdrawal.java
|--- Main
|----- Field.java
|----- FieldValidator.java
|----- Main.java
|----- Menu.java
```

Figure 1. Project File Structure for Project 1 – My Banking System.

UML Class Diagram

Please refer to the attached image together with this document.

Required Functionalities

Your program should be able to do the required functionalities stated below.

For ***Banking Module***, your program should be able to accomplish the following:

1. Creating a New Bank
2. Logging to a Specific Bank (it should fail if credentials are incorrect)
3. Showing Accounts registered under the logged in bank user.
    a. All Accounts
    b. Credit Accounts
    c. Savings Accounts
        i. When creating a new savings account, the initial deposit must be reflected properly on the savings account's balance.
4. Create New Accounts. This should fail if the account number already exists.
    a. Create New Savings Account
    b. Create New Credit Account

For the ***Account Module***, your program should be able to accomplish the following:

1. Account Login. The user must be prompted first about which Bank to log into.

For the ***Savings Account Module,*** your program should be able to accomplish the following:

1. Fund Transfer. (Can only happen between Savings Accounts)
    a. Internal fund transfer, meaning transferring money from another account on the same bank.
    b. External fund transfer, meaning transferring money from another account on different banks. Includes processing fee set by the bank itself.
2. Withdrawal.
3. Deposit.

For the ***Credit Account Module,*** your program should be able to accomplish the following:

1. Credit Payment.
    a. Credit payment can only happen when the target account is a savings account.
2. Credit Compensation.
    a. Pay some amount and reduce loans attributed to this credit account. It should not be greater than the amount of credit currently recorded.

Functionality Restrictions

For all accounts, regardless of type, every successful transaction must be recorded appropriately. An example would be when fund transferring from account to account, the transaction statement should denote to which account the money was transferred from, including the amount of money transferred and which bank (in the form of the bank name) the recipient account was registered from. This should be the case regardless of whether it was external or internal fund transfer.

Limitations and Warnings

You are forbidden in using any function, classes, and interfaces that are not yet discussed in your lecture session.

You are also forbidden in creating additional functions/methods on every file (including the rest of the classes). The signature or definition of the function(s) and/or methods should also not be modified. This includes but not limited to:

- Class attributes and names.
- Function name
- Function parameters and data types
- Function return type

Violating this note will result in significant sanctions imposed by your instructor.

For any student with similar code submissions, ignoring variable/function/class name changes, white spacing, etc., your score will be divided upon how many other students is found to share such code similarity. Since you decide to submit the same output, then it is reasonable that the score will be divided upon each, and everyone involved in such practice.

Submission

You will submit a compressed file containing all the Java (no .javac please) files required for this project. It should also include a text file describing the name of the students partaking in your group. Submit the files on or before March 19, 2024. No due date extension, and no ***failed buzzer beater submission*** will be entertained by your instructors.

The name of the compressed file is:

GroupName_ProjectBanking.rar

For example:

JOSS_ProjectBanking.rar