Project 1
Bank Database

Due March 12th

Summary

Design and implement a simple database system for a bank and provide an API that works with the database.

The “backend”

From a broad perspective, the database consist of a list of accounts. Clients and bank workers and/or administrators can then perform transactions and other actions on those accounts. For this project, we will not worry about authentication: don’t worry about implementing admin users or making sure each client can only access their own account.

Accounts

Each client can have a checking account, a credit card account, or a combination of both - each with a separate and unique account number. However, personal information about the account owner should only be stored once. In other words, there should not be two separate account objects storing the same personal information. Choose an appropriate data structure to represent this idea.

Additional clarification: for example, John Doe owns two accounts in the back, one checking account with account number 1001 and one credit card account 2002. If we were to change the personal information for 1001, then the change should be reflected on account 2002 as well. There should not be a case where 1001’s owner becomes Jonny Doe and 2002 is still owned by John Doe.

Transactions

Clients should be able to perform transactions on their accounts. (Again, don’t worry about authentication. We will assume all the transactions are legitimate.) For example, you can make deposits to the checking account, make purchases with the credit card account and use the checking account to pay back what the credit card account owes.

Other Actions

Bank workers and/or administrators should be able to open, close, and modify accounts, as well as checking all account information and running reports.

The “API”

There should be an interface file for your program that is implemented by some other file that interacts with the database. This interface file should consists of all the transactions and actions supported. In reality, this interface file should be written first, while the database would then “implement” this. You can choose the best approach that suits yourself. Remember to include inline comments or explain in the documentation how each function works.

For example, you will likely need a function that deposits money into checking accounts, then the interface function could look something like

```
String deposit(String accountNumber, double amount);
```

Note the return type is String instead of void because the String can return whether the transaction was successful or not, and if unsuccessful, what was the reason. This is not a requirement but it’s a nice feature to have.

The "frontend"

At this point, you can already have a test class that tests all the functions that are in the interface. Ultimately, we then want a main function that links a file reader class that reads the inputs and translates them to functions calls to the API. Additionally, the program should generate a log file and an output file.

File reader class

It is highly recommended that you have a separate class dedicated to reading the input file line by line.

Inputs

All the transactions/actions should be handled from an input **file** (not the console) where each line corresponds to each action. You will decide your own file format but do provide an explanation in the documentation.

For example, a line in the input file that says

```
NCA JOHN DOE 123000456 500.00
```

This can mean John Doe just opened a new checking account with a $500 initial deposit, his social security number is 123000456.

```
PCC 20812 41998 125.33
```

This can mean pay credit card from account 20812 to 41998 with the amount of $125.33

You have complete free choice to define your own input format and contents. However, I would prefer you to include an input file that will best showcase all the different functionalities of your program.

Log and output files

The log file should contain the list of results for each and every action performed. For example, after a deposit action, the log file should have a line that

that states “deposit successful for x account of y amount”.

The output file should provide a snapshot of the state of the database before the program terminates. This should include all the accounts and their balances. This can either be a part of the main function’s routine or a function supported by the API.

Level of details

The main goal of this project is to piece together a program with all the different components. Therefore, having just the name and SSN (and no Address, Date of birth, e.t.c.) will be completely OK for the personal information. Similarly, in terms of error and exception handling, don’t worry too much about edge cases (such as someone opening an account with negative balance). Having those features would be nice and could yield extra credits but it’s not the main focus of the project.

I would recommend the first version of the project to only include minimal functionality that opens new accounts. Once the first version is completed, you can then incrementally add on other features.

Rubrics/Grading Guide

This project (as well as all the other projects) will be graded “max = 10 point”, then scaled by multiplying 1.5 or 2.0 depending on your performance on the other projects.

(3 points) Account Objects and structure

Using proper OOP structure for all classes

Having necessary set/get functions

Decently realistic logic

(3 points) Interface and implementation

Having an interface file

Have the interface implemented

"Frontend" exclusively using the interface

(3 points) Frontend file I/O

Output file

Log file

Input file

(1 point) Documentation

(? points) Extra credits possible

Exceptional error handling

Robust interface