Final Project: Online Banking System Design Document

Ziyang Sheng, Zimou Sun, Wanzhi Wang
CS611, Lecturer: Christine Papadakis-Kanaris

***Abstract*—This is the design document of our final project. In the project, we built up a banking system that could be used by both customers and managers to operate banking businesses. The document specifically introduces how we designed different parts of our program as well as our thought processes. Hope you will like our work, and wish you a happy winter break.**

I. Work division

The project is completed thanks to the joint effort of all three team members. To be specific, Ziyang is responsible for the front end design including all the user interfaces, as well as the data management for users. Zimou and Wanzhi completed the back end. Zimou constructed most of the basic functions and the database. And Wanzhi did the stock part and most of the design document. We also worked together, collaborated on designing and debugging.

II. Introduction

THE main goal of this project is to create an ATM software with basic functions like saving, withdrawing as well as some extra ones like loan and stock. Both customers and managers could use this software to meet their purposes. Customers are able to create accounts, save/withdraw money, making transactions, loans and invest in stocks; managers could check customer status, view transaction history and the stock market.

In this document, we divide our main design into several parts: mechanics, architecture, database and graphical user interface. In sections 2 to 5, we will show details of our design structure and how it works. Sections 6 is the analysis of our work, followed by section 7, the conclusion. Section 8 is the reference.

December 16, 2021

III. Mechanics Design

In order to decide what mechanics we need to create in our program, we followed the assignment details and came up with functions that need to be covered.

A. User Account

Firstly, there are two groups of users who have totally different objections. And if the user is a customer, he/she is supposed to have his/her own banking accounts. Hence, we link each user to a user account (this is not a banking account!) consists of an ID and a password. In such case, when a user signs in his/her user account, the program will know his/her identity and leads him/her to his/her bank accounts.

We also added the signing up feature (of course!).

B. Bank Account

Each customer has three different types of bank accounts: saving, checking and security. When a new user account is created, there's no bank account linked to it. The user could choose to create one of each type of account for 5 corresponding currency. There could be transactions between accounts (costs 1 corresponding currency), and the user could also deposit to or withdraw from (costs 1 corresponding currency) them. One thing to be noted is that transaction into security account only takes place when the user has over 5000 USD in his/her saving account, and the transactions has to be greater than 1000 USD.

C. Loan

As mentioned in the requirements, customers should be able to request loans from the bank. We added that feature. In order to make it more reasonable, we set that the total loan has to be less than ten times the amount in saving account. Customer will be charged for 5% as an interest when paying his/her debt.

D. Stock

One more function to be added is stock. We allow the customer to invest in stock only if he/she has savings in security account (which means the customer meets the preconditions needed to transfer money into security account). We used an API to grab real-time stocking information online. User could search stocks by putting in the company symbol, and then choose to buy/sell stocks.

Since we are using real online data, there's no way for the manager to change stock volume and price (as required in the assignment details), instead, the manager could view stock information as customers do.

E. Transaction History

If the user is a manager, he/she has the power to view all transactions that take place on the same day.

IV. Architecture Design

Creating a fully functioned banking software requires us to think carefully about the overall structure of our program. We decided to separate the front end and back end to different files to keep the program neat and modifiable. All files ended with ”UI” are front end files which contains the appearance of and interactions between user interfaces. Files that do not contain ”UI” in their names are back end files. All data and underlying logic are contained in those files.

A. Front End Files

Each of the front end file defines one single frame in our GUI. The description of each file is shown below:

- *AccountUI.java* – The user interface for customers to create a new account.
- *BorrowLoanUI.java* – The user interface for customers to apply for a loan.
- *CustomerUI.java* – The user interface for customers to choose an action.
- *DailyReportUI.java* – The user interface for the manager to view daily transaction record.
- *InformationUI.java* – The user interface for customers to view their account information.
- *ManagerUI.java* – The user interface for managers to choose an action.
- *ReturnLoanUI.java* – The user interface for customers to return the loan.
- *SaveUI.java* – The user interface for customers to make a deposit.
- *SignUpUI.java* – The user interface for users to sign up.
- *StockUI.java* – The user interface for the stock business.
- *TransactionUI.java* – The user interface for customers to make transactions.
- *ViewClientUI.java* – The user interface for managers to view customer information.
- *WelcomeUI.java* – The entrance of the whole system and the welcome interface.
- *WithDrawUI.java* – The user interface for customers to withdraw money.

B. Back End Files

First of all, we set up a ATM class that keeps track of the current user. It also defines functions for users to login.

- *ATM.java* – The manager class for both customer and manager login.

The user class keeps track of ID and password, and class that inherits it, customer records the bank accounts a customer has as well as stocks he/she buys.

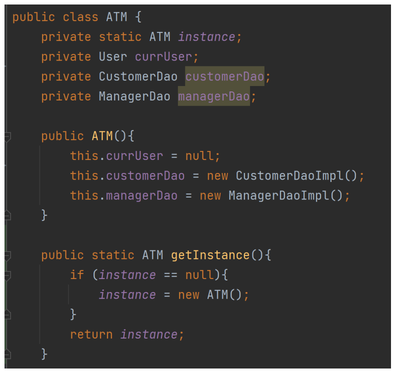
- *User.java* – The abstract class for all kinds of users.
- *Customer.java* – The class in charge of customers.
- *Manager.java* – The class in charge of managers.

We created an abstract class called account and sub-classes for each account type.

- *Account.java* – The abstract class for all types of accounts.
- *AccountType.java* – The enum class for three account types.
- *CheckingAccount.java* – The class for checking account.
- *Currency.java* – The class in charge of all kinds of currency.
- *CurrencyType.java* – The enum class for three currency types.
- *CustomerDao.java* – The Interface for customer DAO.
- *CustomerDaoImpl.java* – The class in charge of the operations on file ”Customer.csv”.
- *Database.java* – The database that stores all the user data in this system.
- *Deposit.java* – The class for customer’s savings.
- *FileUtils.java* – The class for utility functions on csv files.
- *Loan.java* – The class in charge the loan business.
- *Main.java* – The Main class of the whole system.
- *ManagerDao.java* – The Interface for manager DAO.
- *ManagerDaoImpl.java* – The class in charge of the operations on file ”Manager.csv”.
- *SavingAccount.java* – The class for saving account.
- *SecurityAccount.java* – The class for security account.
- *Stock.java* – The class in charge of the stock business.
- *Transaction.java* – The class in charge of all kinds of transactions.
- *User.java* – The abstract class for all kinds of users.

C. Connection

In order for the front end files to receive data/activate functions stored in back end files, we made ATM an singleton class using the singleton pattern, and let front end files to refer to the ATM instance. From there, we made a connection between front end and back end files.



```
public class ATM {
    private static ATM instance;
    private User currUser;
    private CustomerDao customerDao;
    private ManagerDao managerDao;

    public ATM(){
        this.currUser = null;
        this.customerDao = new CustomerDaoImpl();
        this.managerDao = new ManagerDaoImpl();
    }

    public static ATM getInstance(){
        if (instance == null){
            instance = new ATM();
        }
        return instance;
    }
```

Fig. 1. ATM Class

V. Database Design

Database class includes the methods to store/read two types of objects, Customers Object, Transaction Object. The actual implementation incorporates google’s Gson library, so that custom objects can be stored/read to json in a more direct manner. Besides that in static field of Database class, there is a map ¡String,Customer¿, mapping a CustomerName to a corresponding Customer Object. Given that there will be a local UserName.json file ,which contains a Customer Object whenever a Customer create an account(of any type), such a Customer Object can be read from local , and then stored to the map right after login.After that this Customer Object will be global accessible. Similar idea also apply to Transaction. The only difference is that Transaction will not be cached to Database. Transaction Objects generated on the same date will be stored in the same “xxxx-xx-xx.json”(i.e. 2010-10-10.json) . So every time when only Transaction objects are needed , we can easily read Transaction from local file.

VI. Graphical User Interface Design

The user interface is implemented using Java Swing. We created nested interfaces based on the back end logic and took care of every required functions, at the same time taking fancy appearance into account. Besides, the data access object(DAO) design pattern is also included in the login part, where we created two ”csv” files to record the users. To be specific, the data access object is a pattern that provides an abstract interface to the user name and password database. By mapping application calls to the persistence layer, the DAO provides some specific data operations without exposing details of the database[2]. According to the single responsibility principle, DAO pattern separates the methods supporting data access that the application needs, from how these methods are implemented. Specifically, all the methods maintaining the user data are isolated in the ”XXXImpl.java” files.

VII. Future Improvement Analysis

There could still be some improvements in our program. The text area could be beautified; stock can be searched in a more convenient way; customer could create more than one saving/checking accounts. Given the tight time we have during final, we think we’ve done a decent job and decide to make those improvements in later days.

VIII. Conclusion

It was so much fun to work on this assignment and we were proud of what we get at the end. During the process we gained a lot of experience in object oriented design (we have more than 50 files in our program!) as well as team collaborating. We learned how to use java swing; we learned google gson; we even found out how to automatically generate XML diagram! We are confident that we will do better job on software development in the future.

IX. References

[1] https://en.wikipedia.org/wiki/Data_access_object

[2] Stock Data provided by Financial Modeling Prep https://financialmodelingprep.com/developer/docs/

X. Appendix

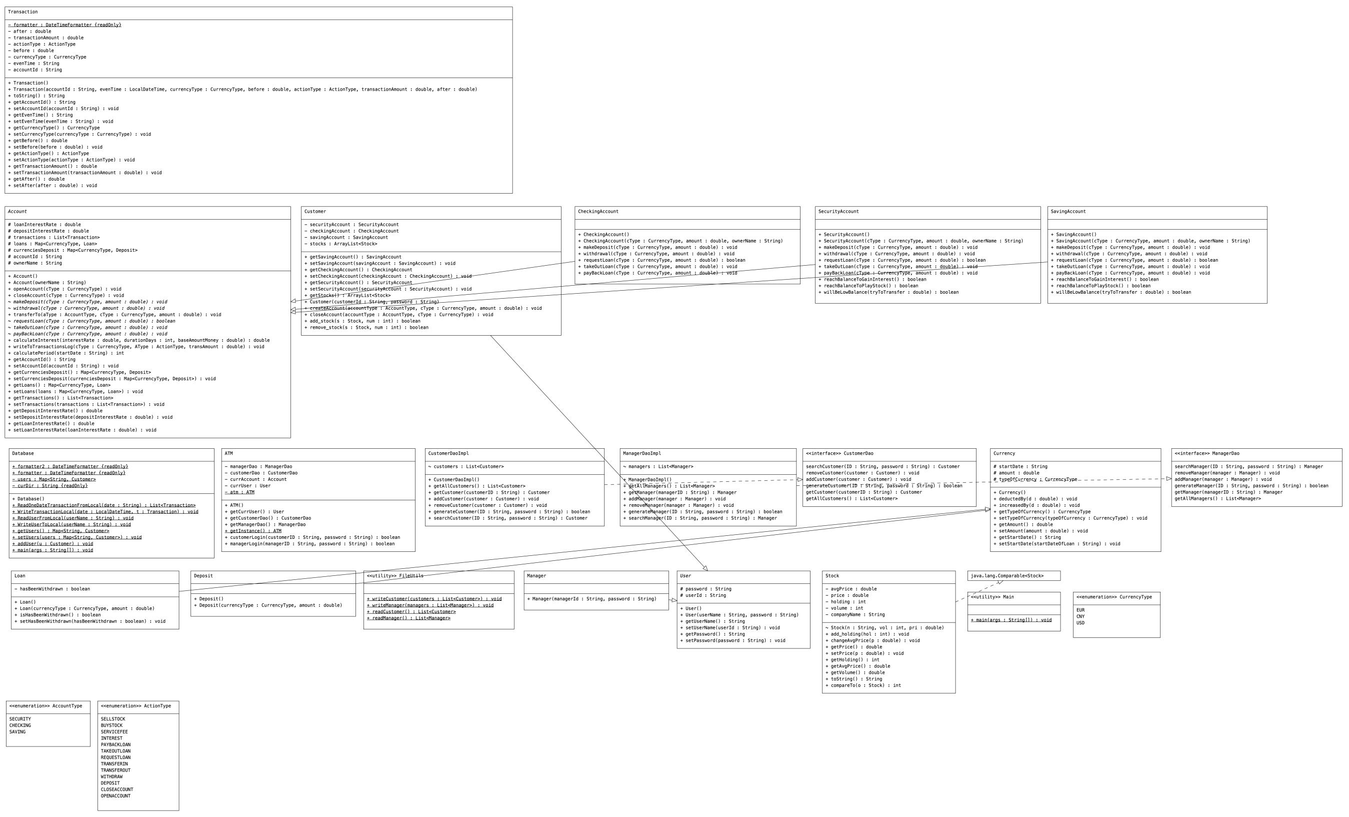


Fig. 2. XML Diagram for Back End Files

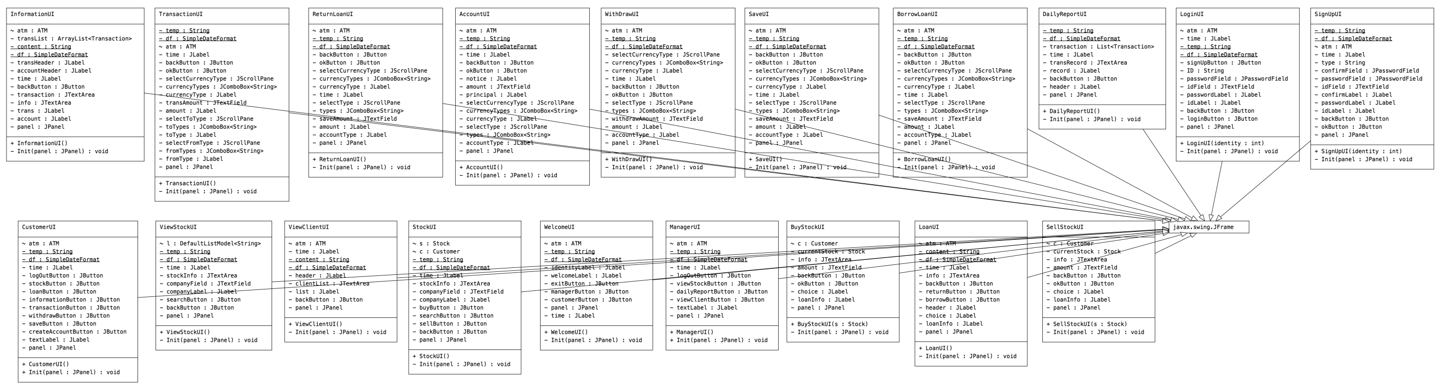


Fig. 3. XML Diagram for Front End Files