CS611 Final Project Design doc

Team 3

Team Members:

- Haijun He (haijunhe@bu.edu github username: CarsonHe)
- Chen Qin (cqin@bu.edu github username: dfjhlsdhf)
- Pengfei Li (lpf00@bu.edu github username: R1cardoo)
- Rhythm Deven Somaiya (rhythm@bu.edu github username: IAmRhy31)

Our github repository link: https://github.com/R1cardoo/bank_final_project

1. Classes we implemented

a) **BankATM**: The main class of our project which provides the entrance to our bank system. If you want to run the program, cd to this java file and enter javac BankATM.java and then java BankATM.

b) **User**: The basic class of class Manager and class Customer. It contains three basic attributes, which is username, password and login status that are common to both customers and manager.

c) **Manager**: This class is extended from class User. Except for username and passwords, this class also set the interest which the manager would pay for customers. A manager role will be created by creating an object of this class. Then manager can get a daily report, check customers who applied for a loan, charge fee and interest from customers and pay interest to customers by calling methods in this class. Manager could also press the button "new Day" to pay and charge interest automatically when a new date comes.

d) **Customer**: Each object of this class represents a customer of the bank. Each object stores the information of all of their accounts and all the transactions. Methods in this class is related to customer operations. When a new customer comes in, create username, password and bank accounts, our program will call methods in this class to save customer information to csv files. Similarly, every time when customers generate a new transaction record, the program will also need call methods in this class to complete the process as well as write new data to the respective csv file in the backend.

e) **CustomerFactory**: We used the factory design pattern in the project architecture, which will be described later in the design document. This class is one part of the design pattern implementation. Once a new customer has completed the registration process, the class will call methods to create a new object of Customer class and save this customer's username and password in the csv file.

f) **Account**: The basic class of all kinds of accounts. We designed three basic attributes for the Account class which is username, type of account and a list of currencies stored in the account. This class implements the basic functionality of a bank account, which is money deposit and money withdraw.

g) **SavingAccount**: It represents a saving account of a customer. Compared to a regular account class, when the customer has a large enough balance in this account, it would call method in the class to add interest to the balance.

h) **CheckingAccount**: Customers could apply for a loan if they have a checking account. This clas represents a checking account of a customer. Each object has an attribute called "loanAmount"

to record the loan that the customer applied before. In this class, we designed two methods to use: loan requesting and loan interest charging, which are corresponding to the operation of customer and bank manager.

i) **SecuritiesAccount**: It represents a security account of a customer. If customer has a balance more than 5000 dollars, he or she is allowed to open a security account. Each object of this class will contain the realized profit, unrealizedprofit and a list of stocks the customer owns. We also implemented two methods used for buying stocks and selling stocks.

j) **Transaction**: This class has 7 attributes: username, account type, date, fee charged, amount, currency type and the transaction's type. When there is a transaction generated, the program would create a new object of this class to represent a transaction record.

k) **Currency**: Each kind currency has two attributes, which are its currency type and value.

l) **CurrencyHelper**: We used singleton design pattern to implement this helper class. The helper is responsible for managing the exchange rate for each currency type. The bank system can call methods in this class to initialize and update the currency exchange rate, and can also call methods to calculate the value of any currency converted to USD. Once a type of currency is no longer supported, the bank system can delete the currency using the method in this class.

m) **Admin**: This class acts as a database, but we used csv files to store data in this project. It implements adding, deleting, modifying and searching data. In addition to the ulnderlying implementation of files data handling, we encapsulated different APIs for the upper level implementation to use according to different operations and file types so that the upper layer can operate the files by directly calling the APIs.

n) **StockMarket**: This class contains all operations on the stock, including updating stocks, adding stocks and removing stocks.

o) **Stock**: Each stock has three attributes, which are its name, id and price.

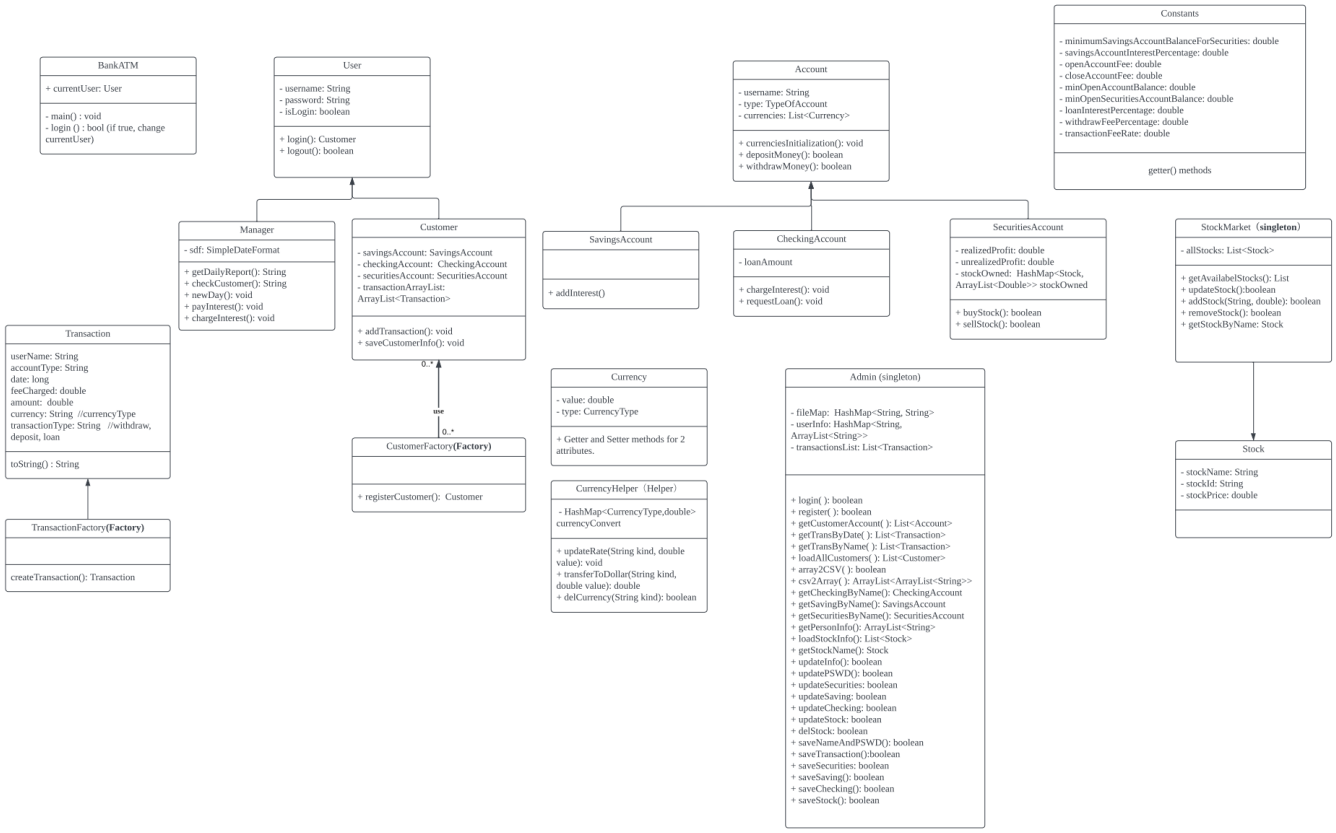
p) **Constant**: Stores constant numbers used in the bank system.

q) **TimeHelper**: This class is used to simulate time. The manager of our application can manually set the time of system to the next day. As he triggered this process, the system automatically pays the interests of customers' savings and charges the interests of customers' loans.

2. Design choices

In our project, we choosed to use two design pattern, which is singleton design pattern and factory design pattern. Admin, CurrencyHelper, StockMarket and TimeHelper, those four classes are all designed as singleton classes. They have only one instance in the whole bank system and this design pattern makes them easy to access. The pattern is useful for coordinating actions across the bank system. We chose to use singleton pattern because these four class above act like tool classes in our project. Each class can easily access these four instances, and once one of the classes call a method to change an attribute value of these instances, the attribute value of all the variables representing this instance will also change.

The implementation of CustomerFactory class is based on the factory design pattern. It is responsible for creating a new customer instance after the registration process. The use of factory pattern allows creating objects without having to specify the exact class of the object that will be created. In our program, we call a factory method implemented by a child class to create an instance rather than by calling a constructor.



3. Object model

Below is our UML diagram.

Fig 1. UML diagram

We also designed six csv files to store the data of our bank system. The design of our csv files has been shown below.

username[key],	password,

Table 1. customer.csv

date[key],	username,accountType,feeCharged, amount, currencyType, transactionType,

Table 2. transaction.csv

username[key],	TypeOfAccount,currency1,currency1,currency2,currency2,currency3,curr3,curr4,curr4,realizedProfit,unrealizedProfit,ownedstockName1,buy price1, count, ownedstockName2,...,

Table 3. securitiesAccount.csv

username[key],	TypeOfAccount,currency1,currency1,currency2,currency2,currency3,curr3,curr4,curr4,

Table 4. savingAccount.csv

username[key],	TypeOfAccount,currency1,currency1,currency2,currency2,currency3,curr3,curr4,curr4, loanAmount,

Table 5. checkingAccount.csv

stockName[key],	stockId, stockPrice,

Table 6. stock.csv

4. Object and GUI relationship

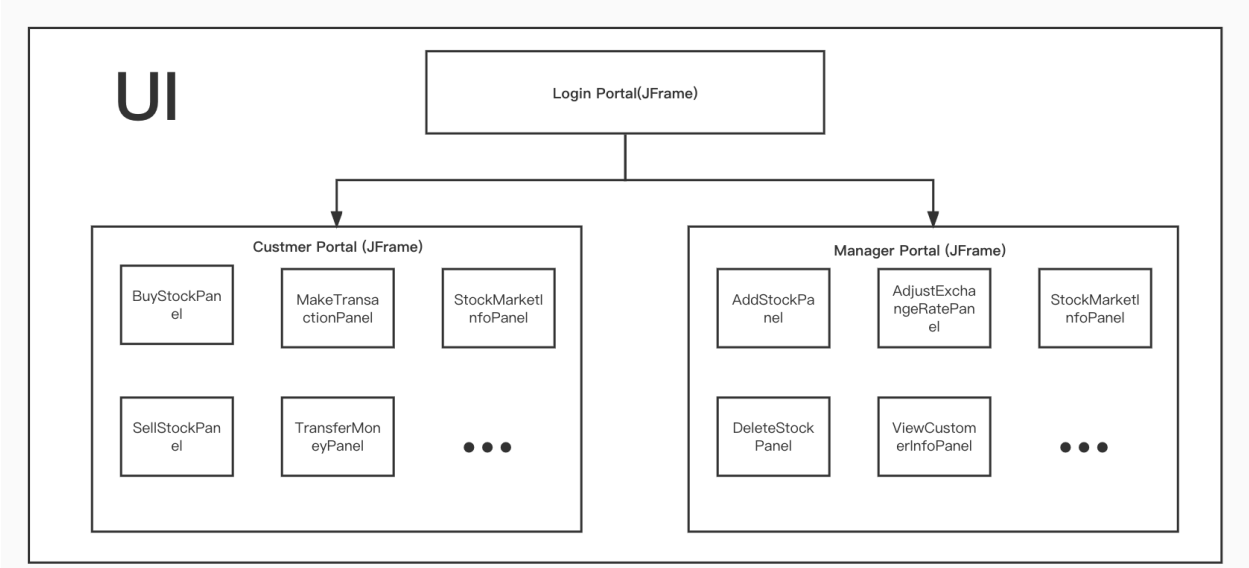


Fig 2. Relationship between object and GUI

Above figure illustrates the relationship between object and GUI.

5. Benefit of our design

First we designed an independent class which is directly used for data storage and data access in our project. The design pattern of this class is singleton pattern, which makes it easy for all other classes in the project to call it, ensures that all values of same attribute in other classes would update the value has been modified, and also makes this module super scalable. In the future, if the program developer wants to add a new file or read a specific data, he or she can edit this class.

For roles in our system, we designed a base class named User. It contains the common attributes for all kinds of roles in our bank system. Both Manager class and Customer class inherit from this base class and then add their own unique attributes and methods. If new roles emerge in the bank system later, developers can create the respective classes by inheriting directly from User class, which reflects the extensibility of our model. The Customer class is created using the factory design pattern. The CustomerFactory class is responsible for customer creating.

Similarly, SavingsAccount class, CheckingAccount class and SecuritiesAccount class all extends from the base class Account. In the future, if the bank could support more types account, developers could create new account class for each type of account through inheriting from class Account.

Below is the structure of our project. We used Java Swing to implement our GUI. The dependencies between each section are shown in the figure. They are both interdependent, but independently responsible for their respective functions, which facilities the modification of the system functions in the future.

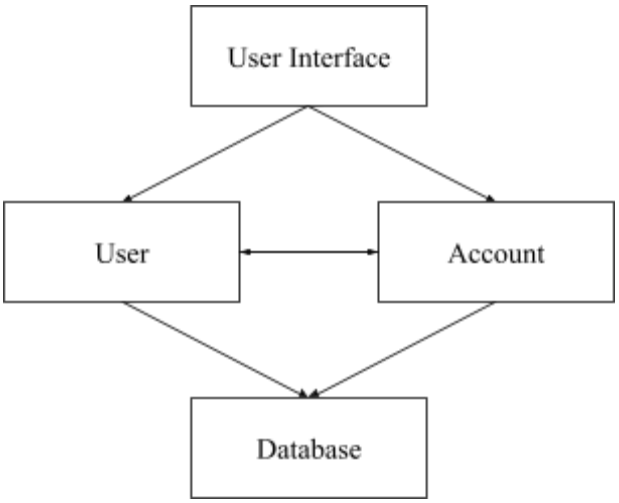


Fig 3. Project structure