611-OnlineBank Design

Yan Tong U31059912
Junyi Huang U57383185
Chenyang Zhang U95109583

#Demand Analysis

According to the given assignment statement, we derived the basic operations(or functions) required by the customer and manager.

Customer:

1. create/close two basic types of account: checking, saving
2. request loan(use collateral)
3. view balance, saving, withdraw, transfer
4. open a security account(if the customer has more than $5000 in his saving account), trade stock through a security account.

Manager:

1. charge fees for account open, transaction, withdraw (automatically executed by the system).
2. approve/refuse the loan request, charge interest for all loans(automatically executed by the system).
3. view the daily transaction report
4. manage the stock market

#Design pattern

By analyzing the specific demand of the assignment, we selected some suitable design patterns after group discussion.

1. MVC Pattern

The bank system is separated into Model, View and Controllers. Model contains the core Back-end code which has all data of the system. View means the GUI of this system(present Model's data to user).Controller is used to listen to events triggered by the View and call methods on the Model. More details in Model/View/Controller can be found in subsequent chapters.

Reason for choosing MVC: Since this assignment required front-end(GUI) and back-end, we think it is natural to use the MVC pattern. There are almost no other simple choices for implementing a system with fancy UI and back-end model. In addition, using MVC pattern provides better scalability for the whole system. Because of the MVC design, we will be able to re-use the existing views to create new views, re-use existing controllers and models to adapt new logic if it is needed in the future.

2. Observer Pattern

It is a tough problem for customers to know that the stock market has been updated by Manager. Here, we find Observer Pattern is extremely useful and suitable in this situation.

Reason for choosing Observer Pattern: We can describe the situation as this: when the manager updates the stock market, this update should be notified to all customers. Here, all customers who have a security account are observers. The Stock Market itself is a subject. By doing this, each time we create a new customer and a new security account, we can simply register the security account to the stock market. As long as the stock market is updated, all accounts will be notified.

3. Factory Pattern

In this bank system, Factories used for creating accounts, customers and collateral. See classes "AccountFactory","CustomerFactory"and"CollateralFactory" for more specific details.

Reason for choosing Factory: Given that there will be many accounts in this bank system and each account should have some private variables(account id, currency,etc.), we think it is natural to set up factories to create objects accordingly. Since creating an account is the most common behavior of customers in the system, the account factory is pretty useful and allows us to create objects more conveniently. Same reason for using customer factories and collateral factories.

4. Singleton Pattern

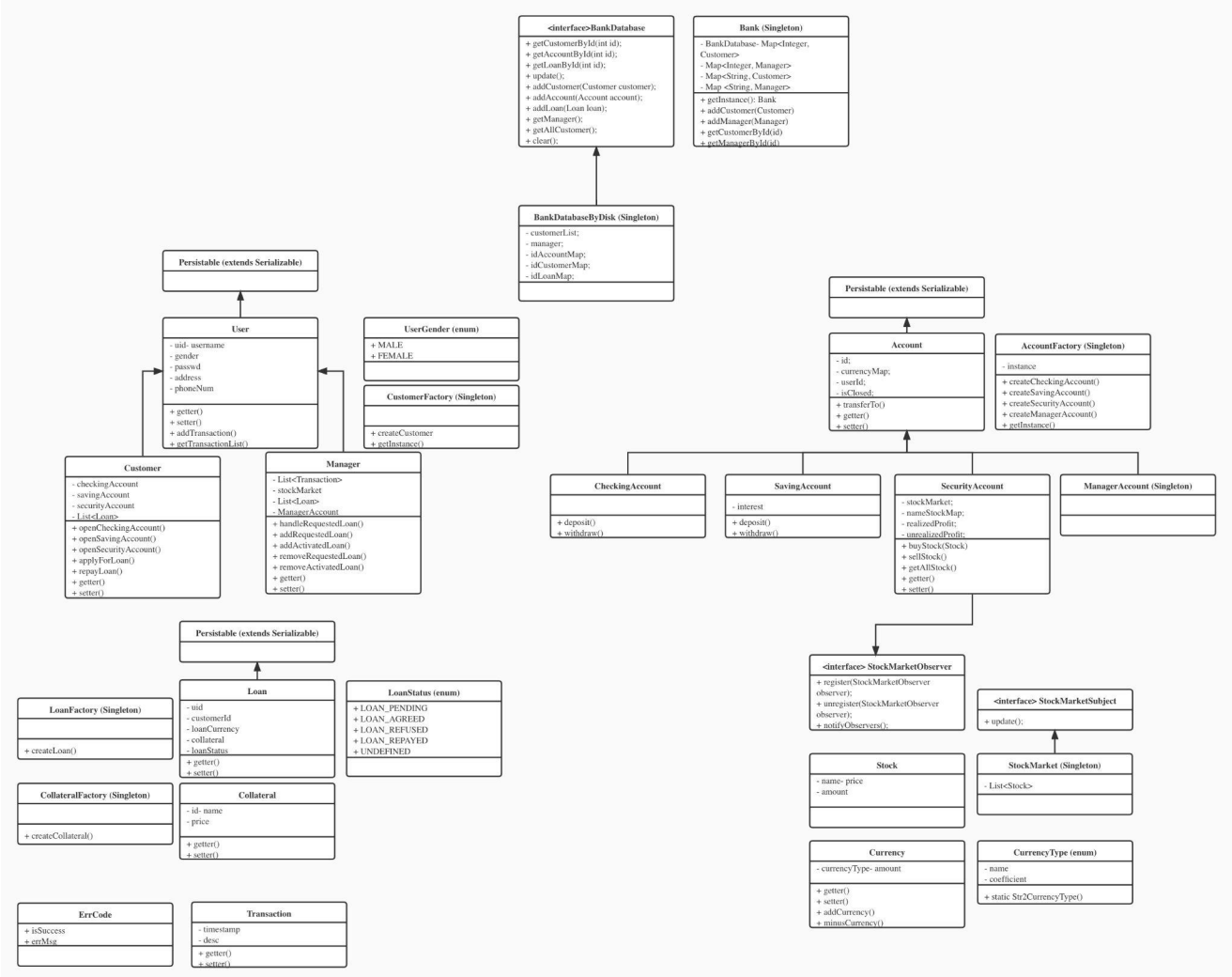
In our design, there are many objects that should be singleton, such as manager, stock market, manager account, all factory objects. Thus it is necessary to use singleton.

#Class structure in Model

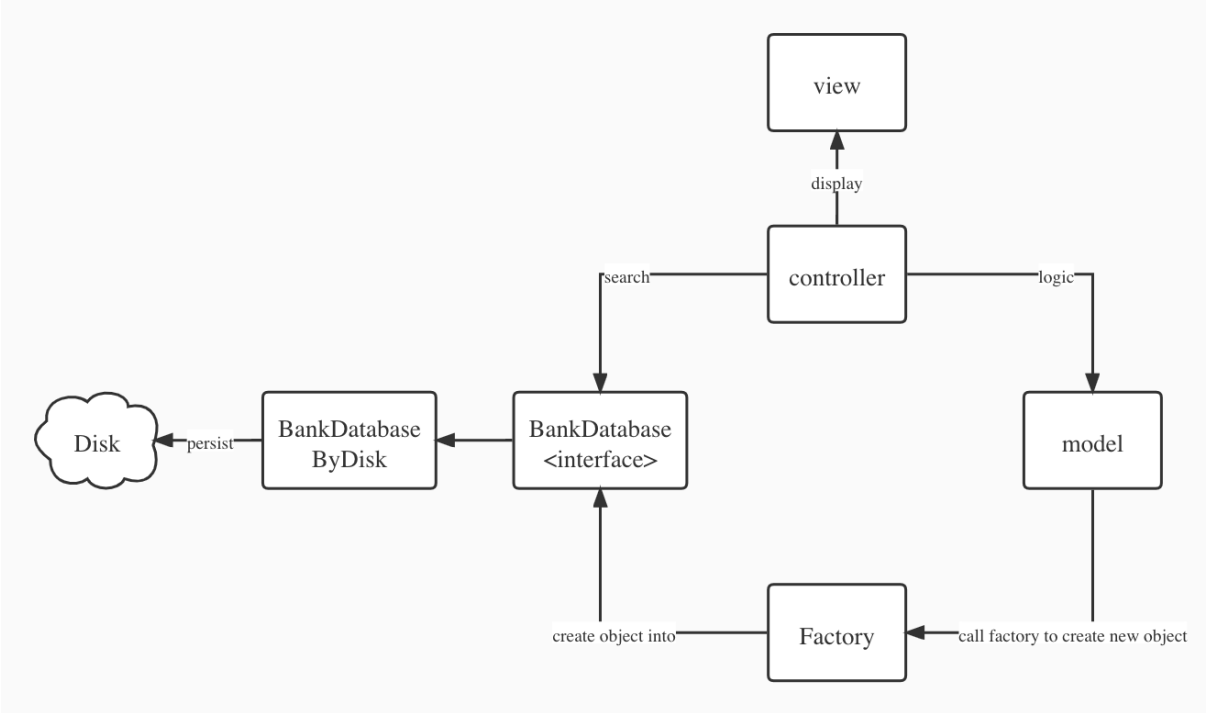
Through the demand analysis and following the OOD principles, we designed some basic classes in the Model(Back-end data). They can be roughly categorized as: bank, account(three types), user(customer and manager), currency, stock, collateral, loan, transaction, database. See the part1 section of this chapter for UML of important classes design in the Model. Explanation for every specific class of Model is in the part2 section of this chapter below.

1. UML for important classes:

OO Design:



System Design:



2. Class Explanation:

Bank

- DataModel for the whole bank. Include manager, BankDatabase, etc. Can add customer to the system, mange the database of the bank system, mange users login.

Account

- DataModel for the account of users. Abstract class. Common variables of all accounts(id, currency, etc.). Include main method of transfer and error hint.

AccountFactory

- Factory class for creating account. Include method for creating Checking, Saving, Security.

CheckingAccount

- DataModel for checking account. Inherited from Account.

SavingAccount

- DataModel for saving account. Inherited from Account.

SecurityAccount

- DataModel for security account. Inherited from Account.

ManagerAccount

- DataModel for manager's account. Inherited from Account.

IdCreator

- DataModel for randomly creating the id of all accounts. Each id means a distinct account. It is the primary key of account.

CurrencyType
- DataModel for different types of currency and their exchange rates. Enum class.

Currency
- DataModel for currency. Include amount and CurrencyType.

Stock
- DataModel for single stock. Include private variables : name, price, amount.

StockMarket
- DataModel for the stock market. Include methods to add, remove, update stock information.

User
- DataModel for users in the bank system. Abstract class. Include common variables of all users' information(name, gender, password, etc.)

Customer
- DataModel for customer. Inherited from User. Include three types of accounts and loan list. Have methods to open/close account, apply loan.

CustomerFactory
- Factory class for creating customers.

Manager
- DataModel for manager. Inherited from User. Include method to get, approve loan request.

Collateral
- DataModel for collateral. Include variables: id, name, price. Customer use it to apply the loan.

CollateralFactory
- Factory class for creating collateral.

Loan
- DataModel for every loan record. Include variables: id, loan price, Collateral, etc.

Transaction
- DataModel for every transfer records. Include variables: from account, to account, amount.

BankDatabase
- Interface of Bank database.

BankDatabaseByDisk
- an implementation of the BankDatabase interface. use a disk as storage.

Persistable
- Interface use to do persistence

ErrCode
- DataModel for error message hint.

StockMarketSubject
- Interface of subject in Observer Pattern. notify observers when stock market updates.

StockMarketObserver
- Interface of observer in Observer Pattern. observe stock market.

#Class structure in View

According to the demand analysis, we already got the needed functions of customer and manager in this bank system. Now after considering these required functions, we can use Java swing to design GUI for our bank system. In the following, we will give all classes explanation in View and the jumping relationships between these interfaces(each class represents a separate interface).

Class & interface Explanation:

Sign
- Interface for user to login. Enter username and password. Click sign in button to Main interface, click sign up button to sign up interface. Screenshot is shown below.

SignUP
- Interface for customer to sign up. Enter personal information of customer. Click confirm, cancel button to back to Sign interface.

CustomerMain
- Home page interface for customer. Click buttons(tree account, log, transaction, etc.) to corresponding interfaces.

OpenAccount
- Interface for customer to open a new account.

Checking
- Interface for customer to manage the checking account. Include balance records, deposit, withdraw, transfer functions. Click transfer button to Transfer interface.

Saving
- Interface for customer to manage the saving account. Same as the Checking.

Security
- Interface for customer to manage the security account. Click transfer, stock button to corresponding interfaces.

Transfer
- Interface for customer to transfer. Enter account id, amount, currency type to confirm the transfer. Checking, Saving interfaces can link to this transfer interface by click transfer button.

TransferInside
- Interface for customer to transfer from his security account to his other accounts. Enter amount, currency type, account type to confirm the transfer.

Stock
- Interface for customer to manage his stocks. Include customer's stock records, stock market records, buy and sell functions.

Loan
- Interface for customer to manage his loans. Include current loan records. Click New Loan button, Repayment button to corresponding interfaces.

RequestLoan
- Interface for customer to request a new loan.

Transaction
- Interface for customer and manager to check transaction records.

ManagerMain
- Home page Interface for manager. Click buttons(transaction, loans, stocks) to corresponding interfaces.

ManageLoan
- Interface for manager to mange loans. Include existing loan records, loan request. Manager can choose loan request to approve.

ManageStock
- Interface for manager to mange stocks. Include active stocks, stock market records. Manger can update the stock market and active/remove single stock.

#Class structure in Controller

After designing the classes in Model and View, now we need to have Controllers to present the Model's data into corresponding interface. By carefully checking the interfaces in View, we determine the needed content(data in Model) that should be displayed in each interface. There are 16 necessary controllers(classes) in total, and we will give their explanations below.

Class&controller Explanation:

SigninController
- Controller class for the Signin UI

SignupController
- Controller class for the Signup UI

CustomerMainController
- Controller class for the home page of customer

LoanController
- Controller class for the customer’s loan management UI

RequestLoanController
- Controller class for the customer to request new loan UI

ManageLoanController
- Controller class for the manager’s loan management UI

ManageStockController
- Controller class for the manager’s loan management UI

ManagerMainController
- Controller class for the home page of manager

OpenAccountController
- Controller class for the customer to open new account UI

CheckingController
- Controller class for the Checking account UI

SavingController
- Controller class for the Saving account UI

SecurityController
- Controller class for the Security account UI

StockController
- Controller class for the customer's stock management UI

TransferController
- Controller class for the customer's transfer UI

TransactionController
- Controller class for the Transaction Log UI

TransferInsideController
- Controller class for the customer's security account transfer UI