Homework: My Fancy Bank

Team Members:

Tiancheng Zhu - U24177199

Ruizhi Jiang - U17637349

Athanasios Filippidis - U95061883

Syahrial Dahler - U42782601

Document Revision History:

Date	Version	Description	Authors
11/11/19	1.0	Initial Draft	Tiancheng, Ruizhi Jiang

Introduction:

1. Purpose:

The purpose of this document is to briefly describe the design and implementation of the project.

2. System Overview:

For this new Bank ATM project, we chose to move forward with Athanasios' backend part and Syahrial's front end part. With Syahrial's front end using a great MVC pattern, we could easily modify and extend it into our final version. With Athanasios' backend has a class structure designed in generic ways, we could smoothly reuse and add some more functionalities in as needed for security account, Stock Market, etc..

3. Design Overview

For the design of this fancy ATM, we use the MVC design model.

Main Design Pattern.

We follow the MVC (Model View Controller) design pattern.

- **Model** - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.
- **View** - View represents the visualization of the data that model contains.
- **Controller** - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps the view and model separate.

DataBase

We are using MySQL database to persist the data. We are choosing MySQL because

1. It is free
2. We are familiar with the implementation

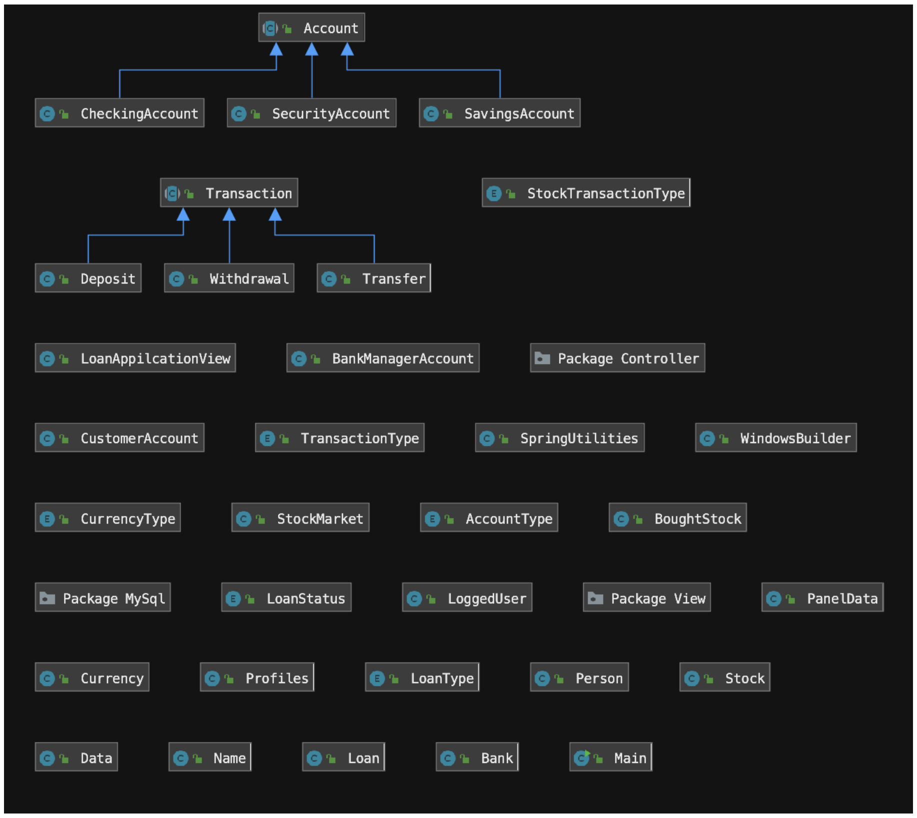
Why do we choose this pattern

1. This pattern makes it very easy to extend the program later when there is new functionality needed without breaking the existing functionality.
   Example: If we need to implement the checking of credit score, we can create a credit score controller, view and model and plug it into our project without changing other components in our code.
2. Because this pattern decouple the main three components, it is much easier and much simpler to reuse and extend each components
3. Each view is and tension of JFrame. So the view can be plugged into a new view that use JFrame

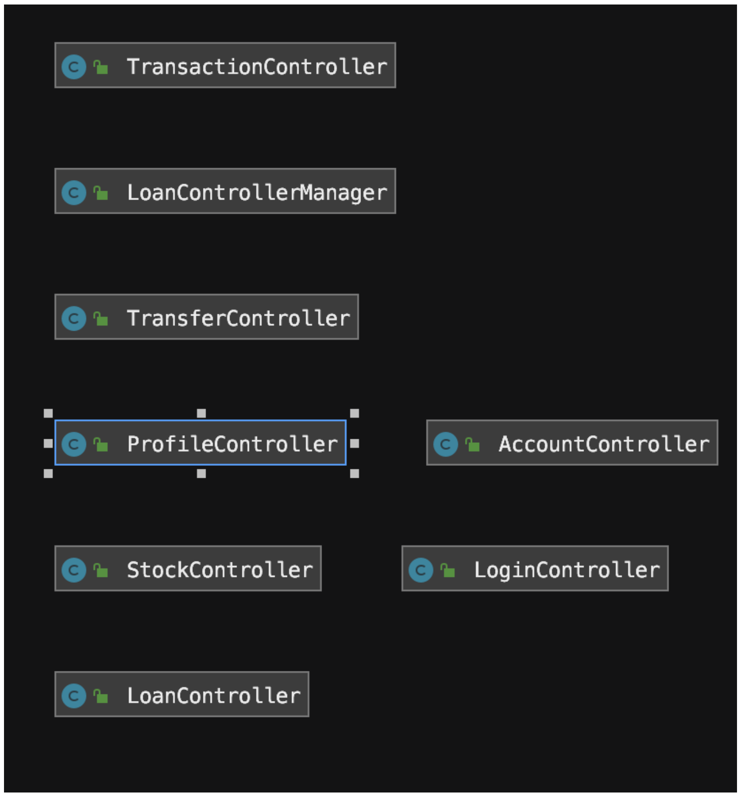
Object-Oriented Design:

1. Class Diagram:

Model:

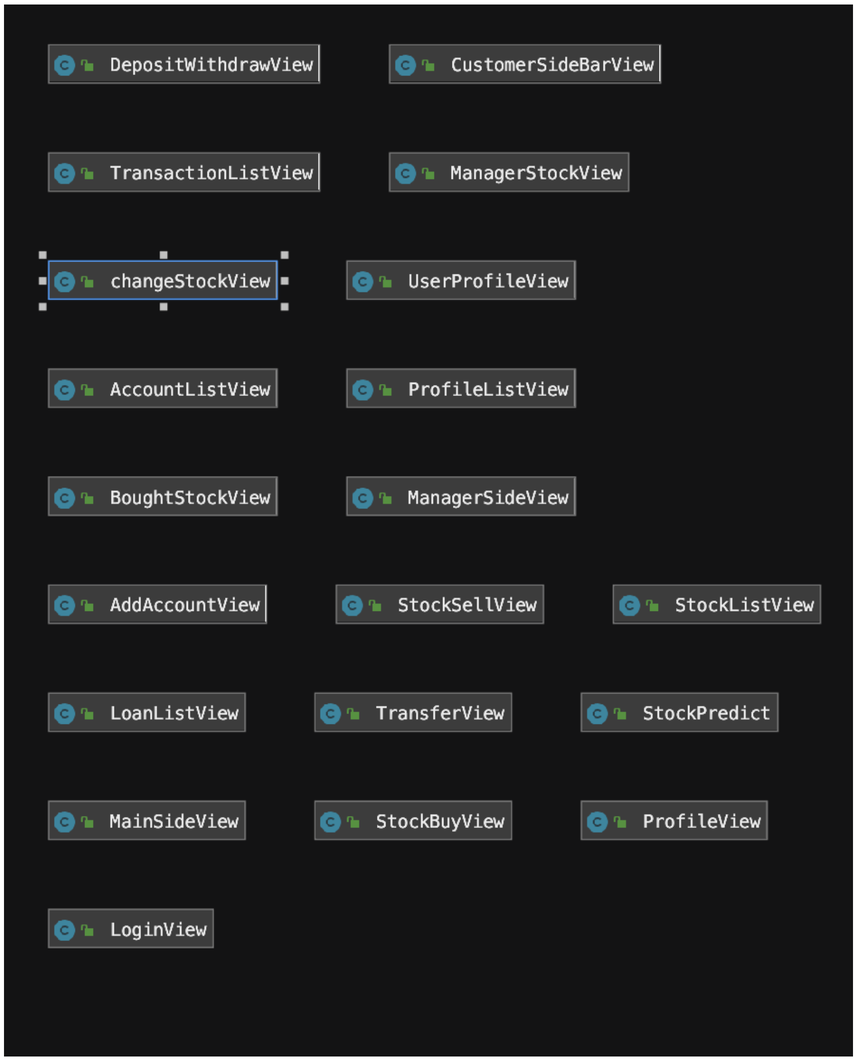


Controller:



DBManager:

View:



2. Class Description:

Model

1. **Account** contain classes of object Account. It is to store the account data such as Account type
2. **AccountType i**s enum class for type of account, Checking and Saving
3. Bank. Class that responsible to register new customers, manager, login and do transaction
4. BankManager. Class that responsible to handle manager process. It is composed of Person
5. BoughtStock. Class to view stocks belong to the customer
6. CheckingAccount. It extends account. Responsible for
7. Currency. Object that store the currency
8. CurrencyType. Enum of currency, USD, JPY, EURO.
9. CustomerAccount. Main class for customers.
10. Data. Static class to load bank and stock sata
11. Deposit extends Transaction. Class to do deposit
12. Loan. Class for creating and applying for loan
13. LoggedUser. Static class to maintain the user that currently logged
14. **Main** is the main class where the main method located
15. Name. Standard object to store name of a person
16. **PanelData**. This is a helper method to update and refresh panels to show the view that is sent by controllers.
17. Person extends Name. contain the information about the customer and manager
18. SavingsAccount. The object that extend Account
19. SecurityAccount. It extends Account
20. SpringUtilities. The class that help to make JFrame in form like position
21. Stock. Class for creation of stocks and processing stocks
22. StockMarket. Class for adding or removing or update stocks to be available for the customer
23. StockTransaction. Enum of type of transaction
24. Transaction. Class to do transaction between one account and another
25. Transfer extends Transaction
26. WindowsBuilder. The first initialization of the GUI
27. Withdrawal extends Transaction

Views (All GUIs)

1. AccountListView**.** The gridview to show the list of account
2. AddAccountView. The view for adding a new account

3. BoughtStockView. The view for customer to view all the stocks he bought
4. changeStockView. The view for the manager to change stocks
5. CustomerSideBarView. Menu bar shown for customer.
6. DepositWithDrawalView. The view for customers to deposit or withdraw
7. LoanListView. The view of all loan. Manager can view all loan, Customer can only see his own loan
8. LoginView. The view for login UI.
9. MainSideView. Main menu bar when nobody logged in yet
10. ManagerSideView. Menu bar for manager
11. ManagerStockView. View for manager to view all stocks
12. ProfileListView. View for manager to see all customer list
13. ProfileView.
14. StockBuyView. View for customer to buy stocks
15. StockListView. View for customers to see his stocks
16. StockPeekView. View for customer to see possible profit/loss
17. StockSellView. View for customer to sell
18. TransactionListView. GridView for manager to see all transactions
19. TransferView. View for customer to transfer funds
20. UserProfileView. View to create and update customer profile

Controllers

1. AccountController. This is the controller that responsible for all things related to account. This controller is responsible in processing data from AccountListView, AddAccountView and DepositWithdrawalView.
2. LoanController. The controller is responsible for, Loan. This controller processes data from LoanApplicationView and LoanListView
3. LoanControllerManager. The controller that manages the loan from the manager's point of view.
4. LoginController. Responsible for all login process. This also define which menu bar to be shown (logged out menu bar (MainSideView) manager menu bar (ManagerSideView), customer menu bar(CustomerSiderBarView))
5. ProfileController. Responsible for Profile process. Inserting new user, and updating existing user information
6. StockController. Responsible for all process regarding Stocks. The controller accessed and processed data from StockLIstView, StockSellView, StockPeekView, ManagerStockView, changeStockView, BoughtStockView
7. TransactionController. Responsible for all transactions. Processed the data for TransactionList View.
8. TransferController. Responsible for all Transfer. Processed the data for TransferView

DataBase Access

DBManager. This class has all methods for crud application from and to database.