Bank ATM Project Report

By:
Bhagyasri Kota(U63334155)
Haoyu Zhang(U81614811)
Qingyuan Zhang(U07172373)

Class Structure:

A brief description of each class and its usage are mentioned in the readme file which is available at the path: https://github.com/bkota0404/611_Final_project
Details of each class are mentioned in the UML diagram given in the report.

Design Choices:

The main idea for the design for this program is to implement a Repository pattern to keep the flow of data structured. Since there are a lot of database queries that need to be executed, the best way to achieve that with minimum interference with other objects is to use this pattern as it increases the reusability of the code, queries as well keep the structure independent of the classes and object definitions. As a result, the project has a very flexible architecture.

Initial analysis for the project object layer was to build it using as many possible independent entities to reduce dependencies and increase reusability with minimal changes. Yet we still wanted it to be connected and the class structure to be meaningful and easy to understand. One such relation is the Stocks object which are used by many of the major objects like Bank Manager and Customers through security accounts. So, we have clearly established a relationship by maintaining the repository which generates the stocks under bank. Manager still can refresh stock prices, add new ones which is achieved by calling an API and updating the repository. Customers will be able to see the stock options that are available to him and purchase them. This purchase model is kept independent since it is for the customer to decide how he/she wants to operate on them and create a relation between the two classes. The most important aspect for the design was to create in such a way it would be user intuitive, and the code can be understood by everyone.

Several other java object oriented structures are utilized in completing the project like usage of Enum classes wherever a predefined set of values are available. For example, we have a predefined set of transaction types in use during account creation, loan request, transaction etc., Interfaces are used wherever there is specific functionality for an entity that can be extended to several others as well.

Objects:

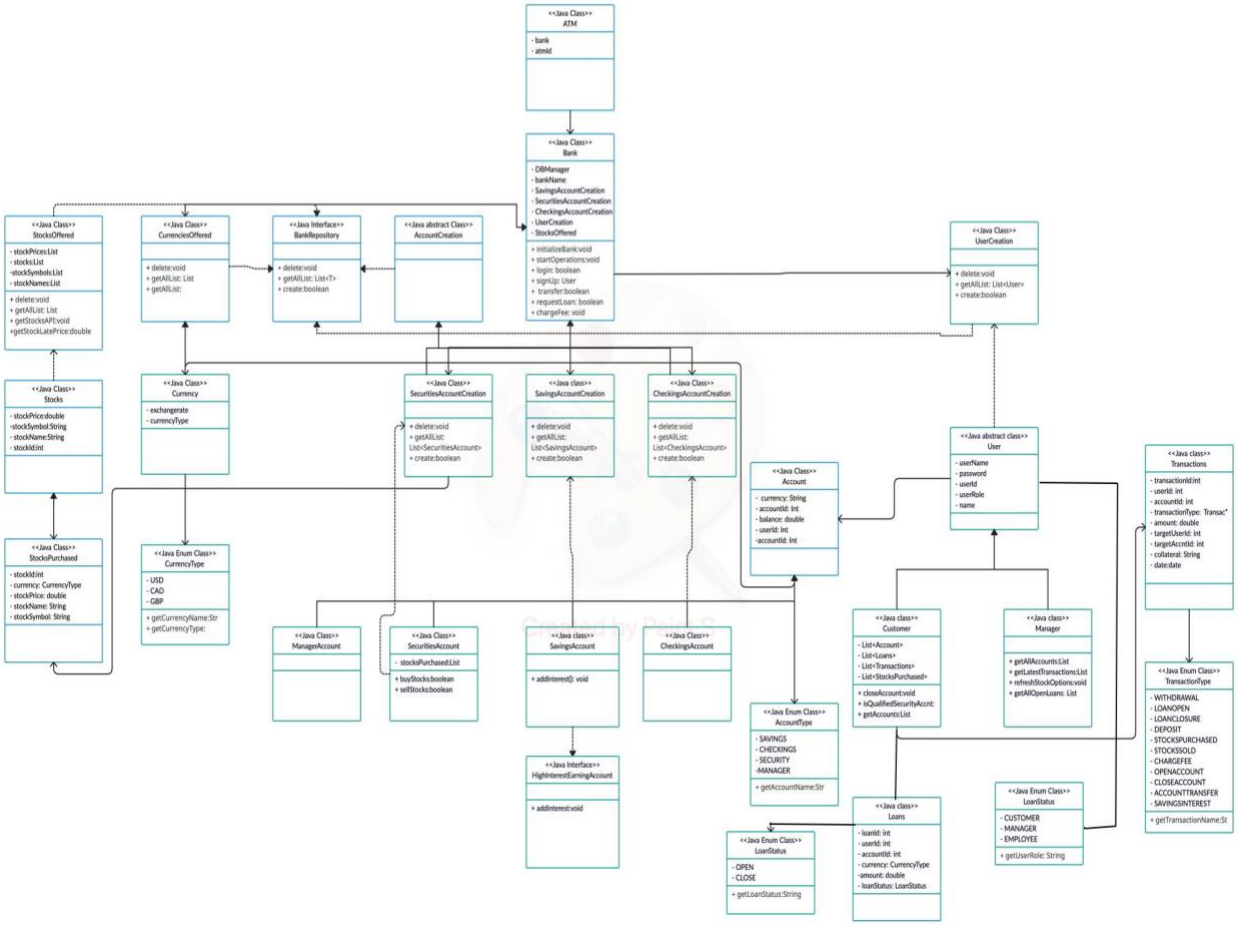
Below are the few important objects used to run a bank in any typical scenario

1. **Accounts**: For the scope of this project, we have 4 different kinds of Accounts which extend the main Accounts class namely SavingsAccount, CheckingsAccount, SecuritiesAccount and ManagerAccount. All these accounts have individual functions and behavior which best explained from UML below
2. **Users**: Here we have two kinds of users; Customers and Manager. Customers must be created during runtime if they wish to use the Banking. Manager will be created by default with certain access. Manager can view and manage accounts, transactions and stocks. Customers can create an Account, transfer money, take loans, purchase stocks etc.
3. **Loans**: The entity is used for customer to take a loan from bank and managers to keep track of all the people who owe money

4. **Transactions**: Transactions is a major object which is used pretty much with all the entities keeping track of actual transactions between accounts, loans and stocks purchased.
5. **Stocks**: Every customer who is eligible to have a securities account can opt to start trading with stocks. Bank maintains a repository of Stocks Offered which are all available for customers to buy and sell. Manager has the ability to drop a stock portfolio as well
6. **Bank**: The most important object available through the ATM project is the Bank. It pretty much handles all the necessary transactions and maintains repositories for all the objects available.
7. **ATM**: The object acts as an interface between the user and Bank to connect customers with the Banking system.
8. Other objects created that help with the project are Currencies and Screens for the GUI

Object Model:

The object model of the project can be very well defined by the UML diagram below which include the classes and their structures, relationships among classes, variables and their definitions, the interfaces implemented by each class. Below is a snapshot of the UML



GUI:

The main idea of GUI is designed for meeting product functions that customers and managers need. We create a login screen for customers and managers to log in, if they have no account, signing up is what they need to do. The screens shown for customers and the manager are similar. We design these screens to extend screen class or item screen class for scalability and extensibility. UI ownable interface is to represent the correlation between different components of GUI, panel especially. In this design pattern, the entire structure is complete and convenient for extending it to a more complicated one.

Also, GUI has plenty of upsides able to enhance users' experience much. The customer screen offers a series of functions for customers, like view their account, view loans they have, view transactions recorded and trade in the stock market. If customers have enough balance in their savings account, the stock market button will be enabled for them to click. The entire screen is of operability. And the manager screen is for managers to check some information stored in the database, like customers, loans etc. Also, the manager will be responsible for the management of the stock market, the screen shows all kinds of information for the manager to manage it more easily.

A couple of screens are designed as an item screen, since there are lists of objects like account list, loan list and so on. We offer dialogs there for customers to create a new account or request a loan, and the screen will refresh in time when the information gets updated. The screen shows information of each object in the database, which would be more convenient for customers and managers to flip through. Meanwhile, each interactive item shown on the screen has corresponding buttons for customers and the manager to handle with.

Benefits:

- **Scalable:** The code is scalable to the point of handle any number of transactions, accounts and users provided we have a database that can handle the data
- **Extendable:** Project is extendable to implement any additional functionalities from as simple as transferring to other bank accounts to making payments to bills etc. It is flexible to the extent that adding new parameters to any classes or objects would involve very minor changes to the code like for example we can associate an email id to a user when he/she wants to sign up and send an email upon successful user creation. The task will be very easy and the design structure we have created will be very intuitive to navigate throughout the program.
- **Reliable**: The code can run for a long duration of period without errors and defects being fixed.
- **Testable**: The functionalities can be tested and run by anyone and everyone and on most versions of operating Systems using right commands
- **Reusability**: The code from the project can be reused for any similar projects or where such functionalities are required. Code such as reading and parsing a file can be used by any projects.

Notes:

The complete documentation and code along with the required drivers and data files are available at: https://github.com/bkota0404/611_Final_project