BankATM Design Doc

## Project Description:

ATM Project is a software programme that works like a real bank ATM system.

## Architecture:

The whole project is encapsulated into two parts: Frontend and Backend. The communication data format between them are small objects which can easily be translated into json.

This design provides flexibility and scalability when swapping frontend gui or deploying the project to different platforms for example internet or mobile devices.

Frontend is driven by Java swing with OOdesign. Backend is made up of two parts: service and database.

Service is made for logic processing and database holding entity states. Model is structured by hierarchy. Each service will create, retrieve, update and delete related objects. For example, Account Service provides functions related to Account, User service provides functions related to User. Services have dependencies among each other. User service will call Account service to build up User objects, and both of them will call Database Service to retrieve data from the database.

The benefit of service + database design brings logic/strategy hot swap in run time, which can provide simple logic programming for operators like bank manager or administrator.

The utilization of databases brings scalability, consistency, and security to data states. Also provide robust index functionality when querying data from the backend services.

SQL db will provide transaction safe ability. During a transfer, if the withdrawal action in the original account is not applicable, further sql statements like deposit to destination account will be aborted. If the deposit function is not able to take balance, the withdrawal on the original account will roll back.

.

## Design Pattern:

*Strategy Pattern* applied for classes that possess common properties will extend from super-class or implement specific interfaces. Inside Account Package, IAccount interface serves as the fundamental interface for other sub-account classes. Transaction is the abstract super class. Other concrete transactional classes extend from it.

*Proxy Pattern* applied in fee deduction. Instead of squeezing the fee charging code into Transaction related function. Fee charging will be an outlier option to functions.

*Singleton* is widely applied in service initialization. While programme is running only exists one service instance

*Iterator Pattern* is used for AccountCollection and StockCollection. Encapsulated the inner hybrid data structure utilization and iteration.

*Factory Pattern* applied in account creation, when client choose type of account they want to create, account factory will create corresponding account instance

*Composition Pattern* applied in multiple currency accounts. Accounts holding multiple currencies have a collection of different currencies.

### Database:

Sqlite Database handles all the data flow in the running programme. Interconnected with database service. Using java getDBmanager to run query, interacting with database by inserting, updating or changing specific request from front end service.