Team 29 Bank Project Design Document

Team Members: Shilpen Patel, Justin DiEmmanuele, George Padavick, and Matthew Gilgo

Table of Contents:

Introduction

The purpose of this assignment was to design, test, and implement an application for a client who wanted to generate income from a virtual Bank. The client referred to as the Bank Manager in the project specification outlined a set of services they wished to offer to customers. These services would be catered to different subsets of customers and were fairly vague in description therefore, certain assumptions were made to ultimately deliver a product for the client. Based on the customers input initially a set of detailed requirements were defined to work towards incorporating into the product. Once requirements were identified and understood the task of designing the product to satisfy the requirements was completed in three parts: Database design, User Interface Design, and Object Oriented Design. The process has been outlined in further detail in the following sections.

Requirements Specification

Before going through the process of design an appropriate Database, User Interface, and Object Design a detailed understanding of the customer requirements was generated. The requirements were generated from the provided assignment specification and through conference call exchanges with the customer. Based on the customers (who will be referred to as the Bank Manager) input the following requirements were generated for the program:

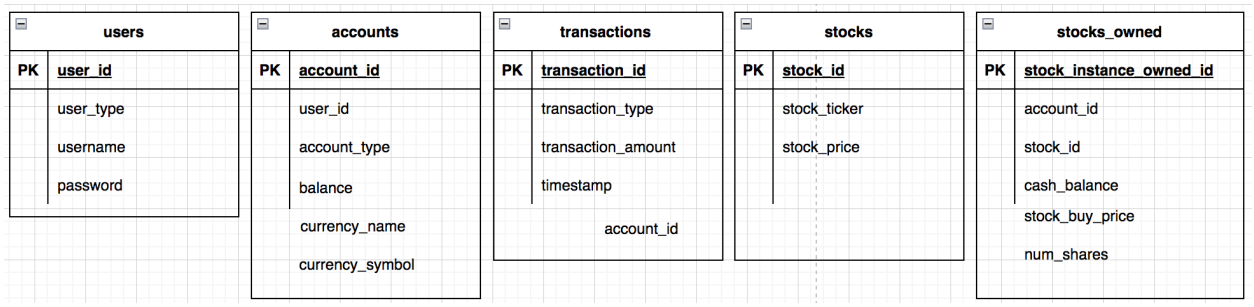
- Generic Requirements:
    - The Bank Manager does not want to interface with customers in person therefore all functionalities of the bank must be implemented virtually
    - The program shall implement a fully digital Bank with certain functionalities detailed below
    - The goal of the bank is to make money by charging fees to customers who are using the banks services
    - Customers and Managers should be able to easily navigate the functionalities of the Bank through an easy to use Graphical User Interface (GUI)
    - The user of the application should be create a user account with a username and password
    - There will be two types of users interfacing with the bank: a Customer and Manager that will each have secure login credentials.
- Customer Requirements:
    - Must be able to create/delete checking, savings, trading, and loan accounts
    - Must be able to manipulate each type of account
    - Checking accounts can be deposited into and withdrawn from
- Manager Requirements:
    - Must be able see who the customers are
    - Must be able to see what transactions the customers have made
    - Must be able to maintain and manipulate the stock market
- Account Requirements:

    - Checking account should operate as a generic cash account where a customer is able to withdraw and deposit (transfer) money between each account
    - Savings accounts can be deposited into and withdrawn from and high balance accounts accumulate interest
    - Trading accounts can be used to buy and sell stock, see current stock portfolio, and see unrealized gains and losses
    - Loan accounts must be charged interest
- Stock Market Requirements:
    - The Bank Manager would like the bank to offer brokerage services to customers who have accounts that satisfy certain metrics. Specifically, the customers who have more than $5000 in their account can open a brokerage account.
    - Manager accounts should be able to update the stock market stocks list and prices for customers
    - Customers with brokerage accounts should be able to trade stocks in the stock market and see realized and unrealized gains
- Persistence Requirement
    - if bank service goes down, their information/money should not go away

From the above requirements outlined above there were three main components needed to be designed and outlined before any implementation could begin. The first component needed was the back-end database design to satisfy the persistence requirements the Bank Manager has requested. The second outline needed was the design and wireframing of the Graphical User Interface based that outlines the user experience with the Bank services. The final and main component of this application was the object-oriented design that implemented all the logic to connect the front-end GUI with the back-end Database.

Database Design

After digesting the requirements an initial design of the back-end database needed to capture the appropriate information for this application. There were five schemas (tables) outlined in the database design that was needed for the application. The tables are as follows:

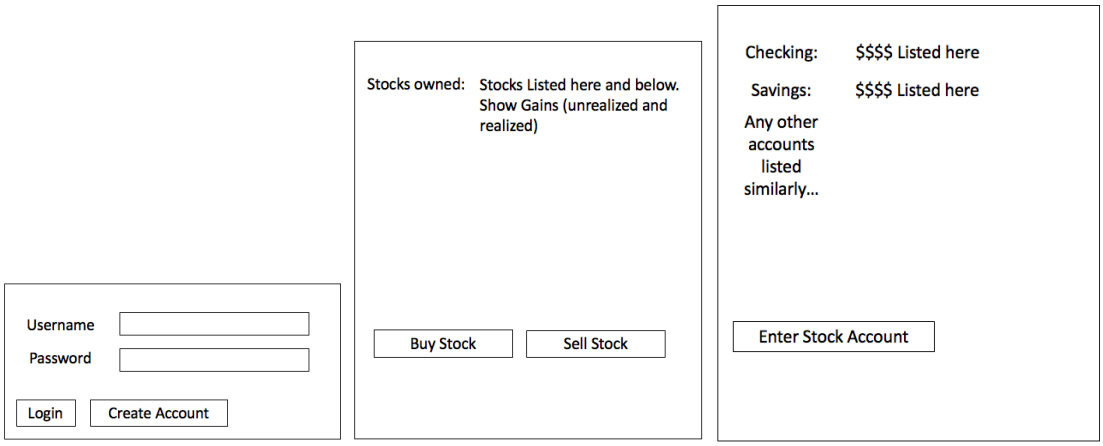


The general idea behind this database design was to have a users table that captures each unique user in the bank with the type of user (Customer or Manager) along with the security credentials. The users once logged in can access the accounts schema through their unique user_id and access their accounts if they

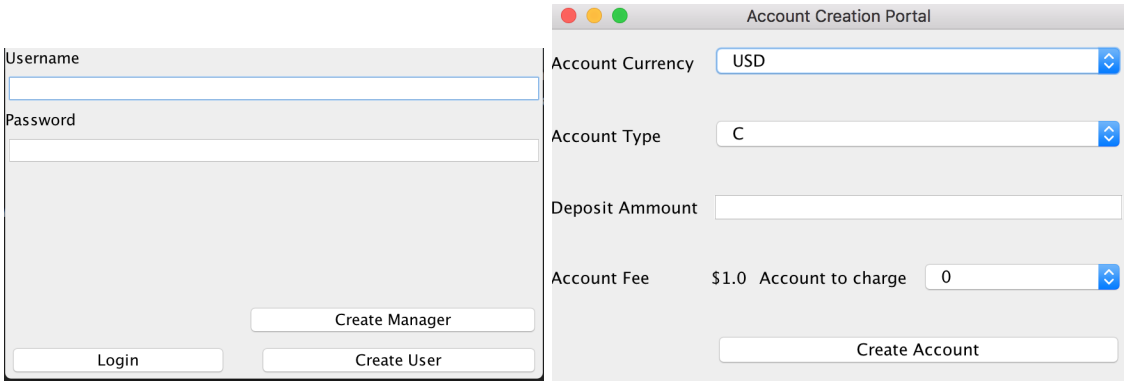
are a customer or access all the users if the user is a manager. The transactions table was needed to capture all the transactions happening in the bank and create a paper-trail of all the transactions in the bank. The stocks and stocks_owned tables are used to store information needed about the stock market, since the stock market is not technically part of the bank they were separated from the other bank tables. Using this structure proved to be extremely useful in implementing the actual application as queries proved to be powerful tools in getting information from the database and coercing it into a form that is easily digestible.

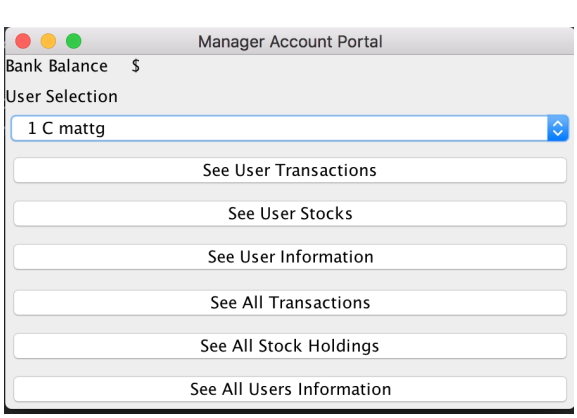
GUI Design

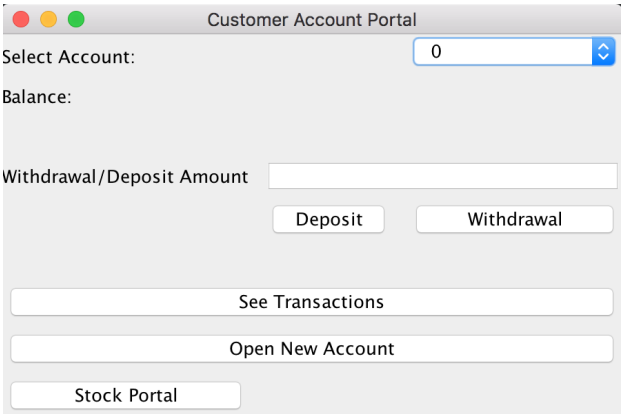
One of the key requirements from the Bank Manager was to allow the user to easily interface with the Bank so that the services offered by the bank could be easily accessed and the information for each type of user could be easily seen. Therefore, before beginning the actual implementation of the application and GUI that would be needed, powerpoint was used to generate initial renderings of screens the user would see and map out the flow of user experience. Specifically, the requirements for each user type were translated into a flow chart of renderings that outlined what the ultimate GUI would look like and feel like to make the ease of use requirement satisfied appropriately. Initial renderings are shown below:

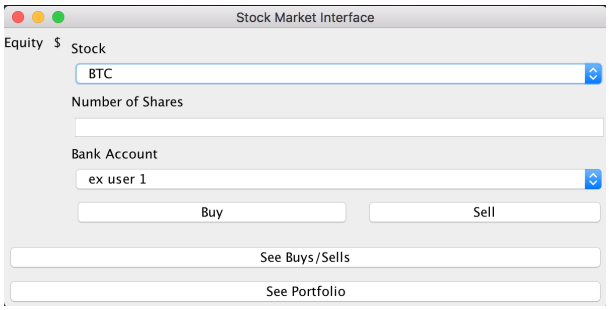


These rough wireframes were then implemented and flushed out upon complete implementation of the Bank application. The actual application GUI screens are shown below:









Manager View of all Accounts:

user_id	user_type	username	password	account_id	account_type	balance	currency_name
1	C	mattg	almostdone	1	CK	1000.0	
1	C	mattg	almostdone	2	Checking	1500.0	USD
1	C	mattg	almostdone	3	Saving	500.0	USD
1	C	mattg	almostdone	4	Loan	100000.0	USD
1	C	mattg	almostdone	5	Loan	100000.0	USD
3	C	shilpentestuser	shilpentestuser	6	Saving	15.0	BTC
3	C	shilpentestuser	shilpentestuser	7	Loan	5.0	ETH

View of all Transactions:

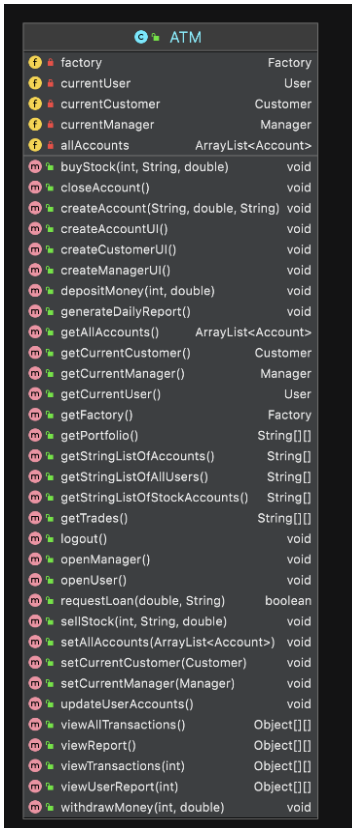
transaction_id	transaction_type	transaction_amount	transaction_time	account_id
1	W	1000.0	1969-12-31 19:00:00.0	1
4	deposit	0.0	1969-12-31 19:00:00.0	1

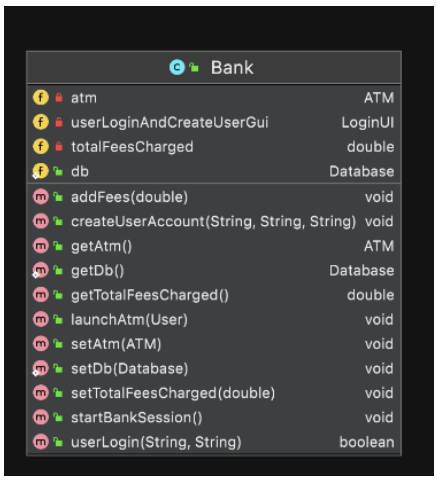
Object Design

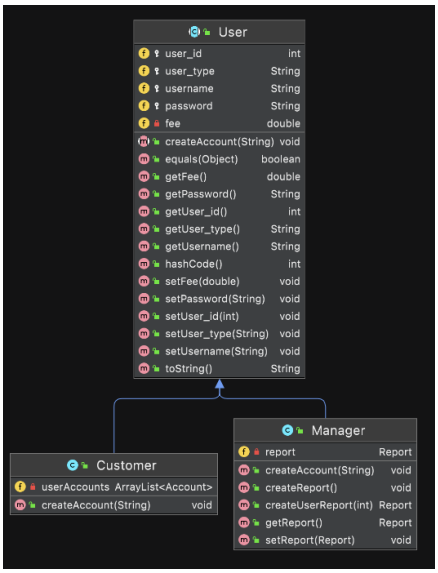
The third component that needed to be designed and implemented to complete the application was the object-oriented design needed to be outlined in detail. As a team the OO-design was done by starting with a clear understanding of the requirements which have been outlined in the requirements specification section of this design document. Once the requirements were understood a detailed UML was created to capture all the necessary objects and connections needed for the logic of the application including the connection between the front-end GUI and the back-end database. In general, the use of design patterns and object-oriented design principles were also thought out during the process of generating the UML diagram. The initial UML had four main components in it, the first was a Bank object that was connected to an ATM object and the appropriate user types (customer/manager) that would be instantiated once a user logged into their account. The second group of objects was an abstract account object with the various account types as children objects. The third major component was based on the transactions that a

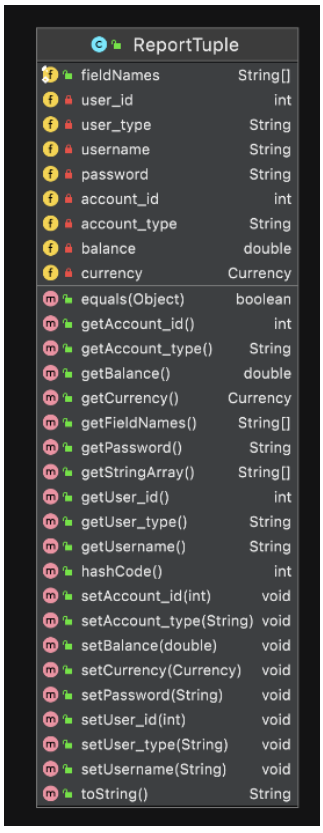
user would be able to enact as part of the services suite the bank offered. The transactions class acted as a parent class for all types of transactions to stem from in the application. The fourth major component was the stock market object which captured the external to the bank stock market that the Bank Manager requested certain customers would have access to trade. The UML components are shown below:

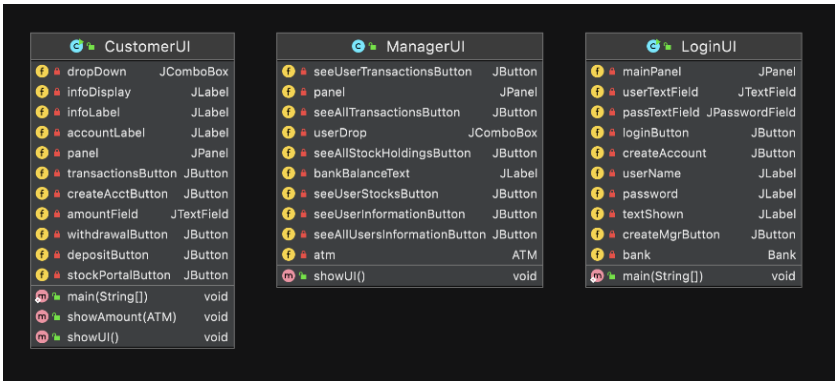
Component 1: Bank, ATM, and User


ATM
factory Factory
currentUser User
currentCustomer Customer
currentManager Manager
allAccounts ArrayList<Account>
buyStock(int, String, double) void
closeAccount() void
createAccount(String, double, String) void
createAccountUI() void
createCustomerUI() void
createManagerUI() void
depositMoney(int, double) void
generateDailyReport() void
getAllAccounts() ArrayList<Account>
getCurrentCustomer() Customer
getCurrentManager() Manager
getCurrentUser() User
getFactory() Factory
getPortfolio() String[][]
getStringListOfAccounts() String[]
getStringListOfAllUsers() String[]
getStringListOfStockAccounts() String[]
getTrades() String[][]
logout() void
openManager() void
openUser() void
requestLoan(double, String) boolean
sellStock(int, String, double) void
setAllAccounts(ArrayList<Account>) void
setCurrentCustomer(Customer) void
setCurrentManager(Manager) void
updateUserAccounts() void
viewAllTransactions() Object[][]
viewReport() Object[][]
viewTransactions(int) Object[][]
viewUserReport(int) Object[][]
withdrawMoney(int, double) void

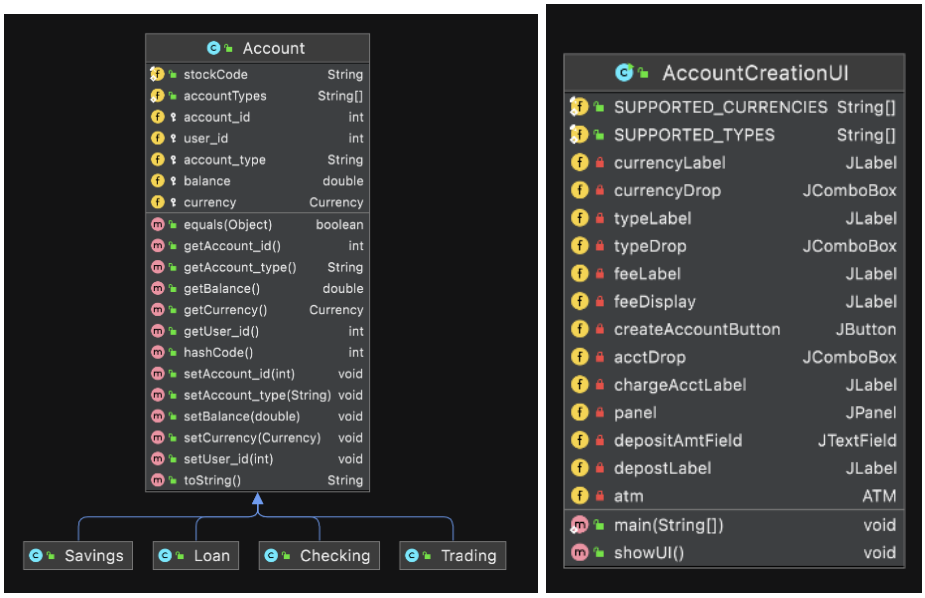

Bank
atm ATM
userLoginAndCreateUserGui LoginUI
totalFeesCharged double
db Database
addFees(double) void
createUserAccount(String, String, String) void
getAtm() ATM
getDb() Database
getTotalFeesCharged() double
launchAtm(User) void
setAtm(ATM) void
setDb(Database) void
setTotalFeesCharged(double) void
startBankSession() void
userLogin(String, String) boolean


User
user_id int
user_type String
username String
password String
fee double
createAccount(String) void
equals(Object) boolean
getFee() double
getPassword() String
getUser_id() int
getUser_type() String
getUsername() String
hashCode() int
setFee(double) void
setPassword(String) void
setUser_id(int) void
setUser_type(String) void
setUsername(String) void
toString() String
Customer
userAccounts ArrayList<Account>
createAccount(String) void
Manager
report Report
createAccount(String) void
createReport() void
createUserReport(int) Report
getReport() Report
setReport(Report) void

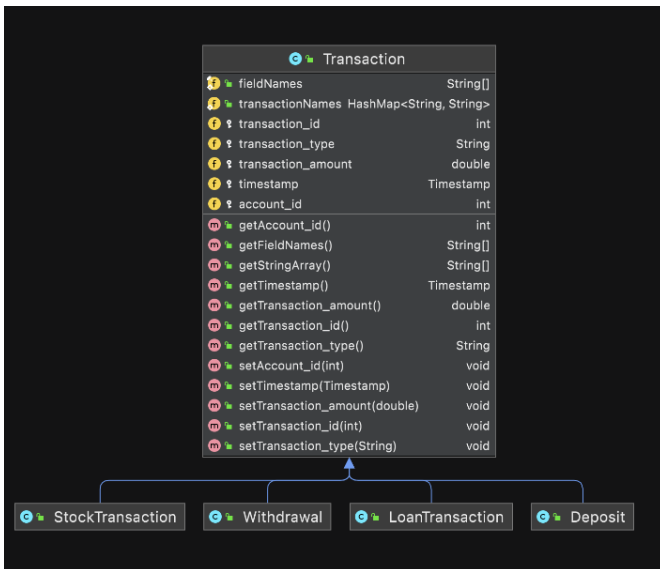

ReportTuple
fieldNames String[]
user_id int
user_type String
username String
password String
account_id int
account_type String
balance double
currency Currency
equals(Object) boolean
getAccount_id() int
getAccount_type() String
getBalance() double
getCurrency() Currency
getFieldNames() String[]
getPassword() String
getStringArray() String[]
getUser_id() int
getUser_type() String
getUsername() String
hashCode() int
setAccount_id(int) void
setAccount_type(String) void
setBalance(double) void
setCurrency(Currency) void
setPassword(String) void
setUser_id(int) void
setUser_type(String) void
setUsername(String) void
toString() String


CustomerUI
dropDown JComboBox
infoDisplay JLabel
infoLabel JLabel
accountLabel JLabel
panel JPanel
transactionsButton JButton
createAcctButton JButton
amountField JTextField
withdrawalButton JButton
depositButton JButton
stockPortalButton JButton
main(String[]) void
showAmount(ATM) void
showUI() void
ManagerUI
seeUserTransactionsButton JButton
panel JPanel
seeAllTransactionsButton JButton
userDrop JComboBox
seeAllStockHoldingsButton JButton
bankBalanceText JLabel
seeUserStocksButton JButton
seeUserInformationButton JButton
seeAllUsersInformationButton JButton
atm ATM
showUI() void
LoginUI
mainPanel JPanel
userTextField JTextField
passTextField JPasswordField
loginButton JButton
createAccount JButton
userName JLabel
password JLabel
textShown JLabel
createMgrButton JButton
bank Bank
main(String[]) void

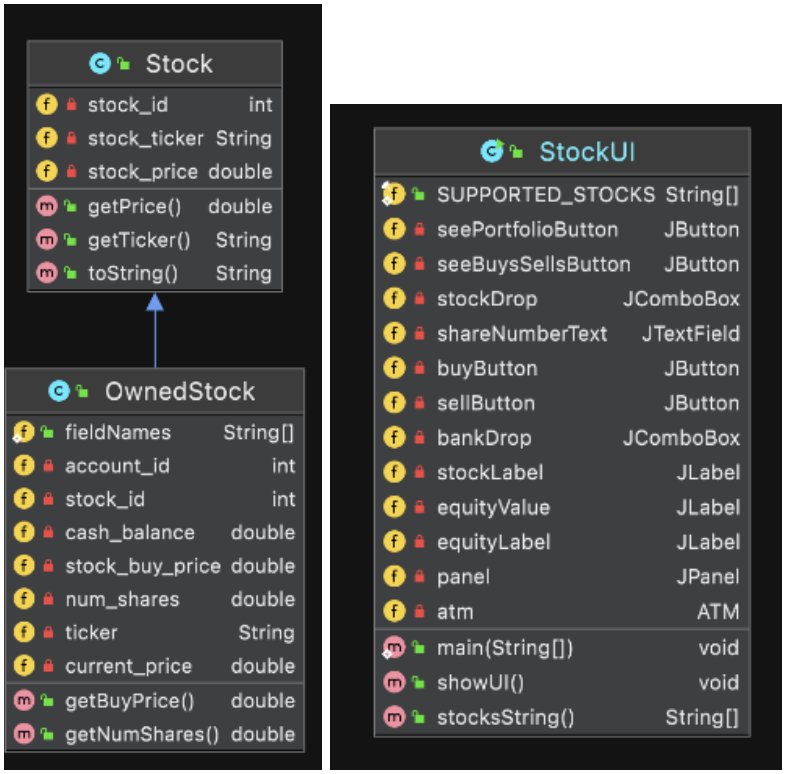
Component 2: Account/Account Types



Component 3: Transactions



Component 4: Stock Market

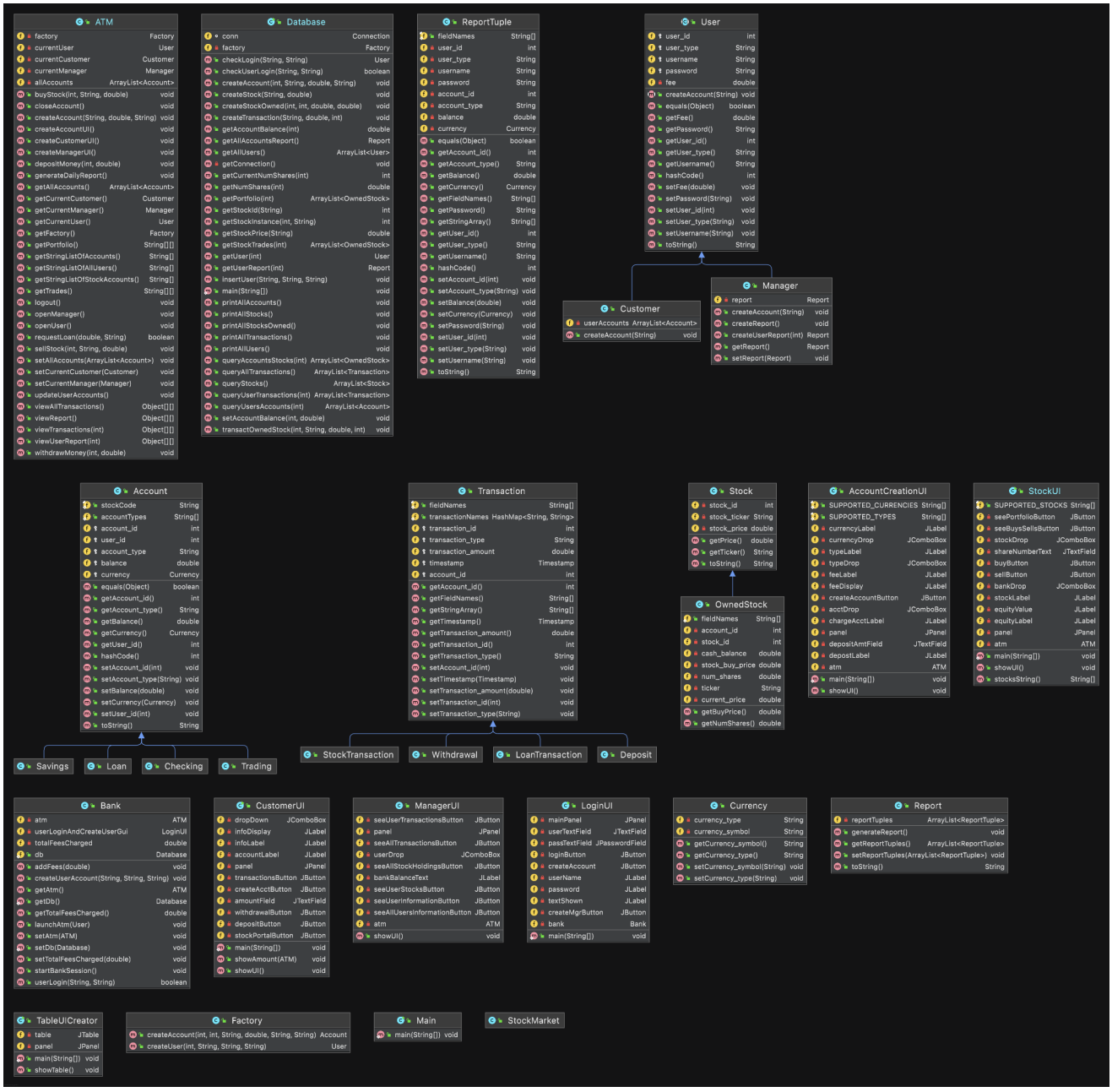


The complete UML is provided in the Appendix.

Conclusions

Overall, the end-to-end design and implementation of this Bank application that was requested by the Bank Manager proved to be a challenging tasks. Due to significant time constraints some of the requested functionalities were not implemented. Specifically, due to poor database design there is no way to capture the amount of money the Bank Manager has made from customers. Furthermore, the application does not have all of the requested constraints on accounts to differentiate which accounts have access to certain services offered by the Bank. Given fewer time constraints and a slight addition to the database all of the functionalities could be easily implemented especially due to the rigorous flexible structure of the database design, GUI design, and OO-Design the application can be easily extended to fulfill all the requested requirements.

Appendix


ATM
factory Factory
currentUser User
currentCustomer Customer
currentManager Manager
allAccounts ArrayList<Account>
buyStock(int, String, double) void
closeAccount() void
createAccount(String, double, String) void
createAccountUI() void
createCustomerUI() void
createManagerUI() void
depositMoney(int, double) void
generateDailyReport() void
getAllAccounts() ArrayList<Account>
getCurrentCustomer() Customer
getCurrentManager() Manager
getCurrentUser() User
getFactory() Factory
getPortfolio() String[][]
getStringListOfAccounts() String[]
getStringListOfAllUsers() String[]
getStringListOfStockAccounts() String[]
getTrades() String[][]
logout() void
openManager() void
openUser() void
requestLoan(double, String) boolean
sellStock(int, String, double) void
setAllAccounts(ArrayList<Account>) void
setCurrentCustomer(Customer) void
setCurrentManager(Manager) void
updateUserAccounts() void
viewAllTransactions() Object[][]
viewReport() Object[][]
viewTransactions(int) Object[][]
viewUserReport(int) Object[][]
withdrawMoney(int, double) void
Database
conn Connection
factory Factory
checkLogin(String, String) User
checkUserLogin(String, String) boolean
createAccount(int, String, double, String) void
createStock(String, double) void
createStockOwned(int, int, double, double) void
createTransaction(String, double, int) void
getAccountBalance(int) double
getAllAccountsReport() Report
getAllUsers() ArrayList<User>
getConnection() void
getCurrentNumShares(int) int
getNumShares(int) double
getPortfolio(int) ArrayList<OwnedStock>
getStockId(String) int
getStockInstance(int, String) int
getStockPrice(String) double
getStockTrades(int) ArrayList<OwnedStock>
getUser(int) User
getUserReport(int) Report
insertUser(String, String, String) void
main(String[]) void
printAllAccounts() void
printAllStocks() void
printAllStocksOwned() void
printAllTransactions() void
printAllUsers() void
queryAccountsStocks(int) ArrayList<OwnedStock>
queryAllTransactions() ArrayList<Transaction>
queryStocks() ArrayList<Stock>
queryUserTransactions(int) ArrayList<Transaction>
queryUsersAccounts(int) ArrayList<Account>
setAccountBalance(int, double) void
transactOwnedStock(int, String, double, int) void
ReportTuple
fieldNames String[]
user_id int
user_type String
username String
password String
account_id int
account_type String
balance double
currency Currency
equals(Object) boolean
getAccount_id() int
getAccount_type() String
getBalance() double
getCurrency() Currency
getFieldNames() String[]
getPassword() String
getStringArray() String[]
getUser_id() int
getUser_type() String
getUsername() String
hashCode() int
setAccount_id(int) void
setAccount_type(String) void
setBalance(double) void
setCurrency(Currency) void
setPassword(String) void
setUser_id(int) void
setUser_type(String) void
setUsername(String) void
toString() String
User
user_id int
user_type String
username String
password String
fee double
createAccount(String) void
equals(Object) boolean
getFee() double
getPassword() String
getUser_id() int
getUser_type() String
getUsername() String
hashCode() int
setFee(double) void
setPassword(String) void
setUser_id(int) void
setUser_type(String) void
setUsername(String) void
toString() String
Customer
userAccounts ArrayList<Account>
createAccount(String) void
Manager
report Report
createAccount(String) void
createReport() void
createUserReport(int) Report
getReport() Report
setReport(Report) void
Account
stockCode String
accountTypes String[]
account_id int
user_id int
account_type String
balance double
currency Currency
equals(Object) boolean
getAccount_id() int
getAccount_type() String
getBalance() double
getCurrency() Currency
getUser_id() int
hashCode() int
setAccount_id(int) void
setAccount_type(String) void
setBalance(double) void
setCurrency(Currency) void
setUser_id(int) void
toString() String
Savings
Loan
Checking
Trading
Transaction
fieldNames String[]
transactionNames HashMap<String, String>
transaction_id int
transaction_type String
transaction_amount double
timestamp Timestamp
account_id int
getAccount_id() int
getFieldNames() String[]
getStringArray() String[]
getTimestamp() Timestamp
getTransaction_amount() double
getTransaction_id() int
getTransaction_type() String
setAccount_id(int) void
setTimestamp(Timestamp) void
setTransaction_amount(double) void
setTransaction_id(int) void
setTransaction_type(String) void
StockTransaction
Withdrawal
LoanTransaction
Deposit
Stock
stock_id int
stock_ticker String
stock_price double
getPrice() double
getTicker() String
toString() String
OwnedStock
fieldNames String[]
account_id int
stock_id int
cash_balance double
stock_buy_price double
num_shares double
ticker String
current_price double
getBuyPrice() double
getNumShares() double
AccountCreationUI
SUPPORTED_CURRENCIES String[]
SUPPORTED_TYPES String[]
currencyLabel JLabel
currencyDrop JComboBox
typeLabel JLabel
typeDrop JComboBox
feeLabel JLabel
feeDisplay JLabel
createAccountButton JButton
acctDrop JComboBox
chargeAcctLabel JLabel
panel JPanel
depositAmtField JTextField
depostLabel JLabel
atm ATM
main(String[]) void
showUI() void
StockUI
SUPPORTED_STOCKS String[]
seePortfolioButton JButton
seeBuysSellsButton JButton
stockDrop JComboBox
shareNumberText JTextField
buyButton JButton
sellButton JButton
bankDrop JComboBox
stockLabel JLabel
equityValue JLabel
equityLabel JLabel
panel JPanel
atm ATM
main(String[]) void
showUI() void
stocksString() String[]
Bank
atm ATM
userLoginAndCreateUserGui LoginUI
totalFeesCharged double
db Database
addFees(double) void
createUserAccount(String, String, String) void
getAtm() ATM
getDb() Database
getTotalFeesCharged() double
launchAtm(User) void
setAtm(ATM) void
setDb(Database) void
setTotalFeesCharged(double) void
startBankSession() void
userLogin(String, String) boolean
CustomerUI
dropDown JComboBox
infoDisplay JLabel
infoLabel JLabel
accountLabel JLabel
panel JPanel
transactionsButton JButton
createAcctButton JButton
amountField JTextField
withdrawalButton JButton
depositButton JButton
stockPortalButton JButton
main(String[]) void
showAmount(ATM) void
showUI() void
ManagerUI
seeUserTransactionsButton JButton
panel JPanel
seeAllTransactionsButton JButton
userDrop JComboBox
seeAllStockHoldingsButton JButton
bankBalanceText JLabel
seeUserStocksButton JButton
seeUserInformationButton JButton
seeAllUsersInformationButton JButton
atm ATM
showUI() void
LoginUI
mainPanel JPanel
userTextField JTextField
passTextField JPasswordField
loginButton JButton
createAccount JButton
userName JLabel
password JLabel
textShown JLabel
createMgrButton JButton
bank Bank
main(String[]) void
Currency
currency_type String
currency_symbol String
getCurrency_symbol() String
getCurrency_type() String
setCurrency_symbol(String) void
setCurrency_type(String) void
Report
reportTuples ArrayList<ReportTuple>
generateReport() void
getReportTuples() ArrayList<ReportTuple>
setReportTuples(ArrayList<ReportTuple>) void
toString() String
TableUICreator
table JTable
panel JPanel
main(String[]) void
showTable() void
Factory
createAccount(int, int, String, double, String, String) Account
createUser(int, String, String, String) User
Main
main(String[]) void
StockMarket