# ATM

# User Interface Functions

Following is a list of the user interface functions we have implemented.

## After Demo We Improved These Parts

We have implemented all the classes and methods in OOP, but fails to debug and run the GUI during demo.

### Manager Side

* `Manage Stock`: It allows the manager to manage stock information in the market. The manager could change price/availability of the stocks currently listing in the market, and they also could add a new stock to the market. Error checking: cannot add same stock names; price cannot be lower or equal to 0.

* `Manager Daily Income`: It allows the manager to check the total daily income, which includes loan interest and account fee.

* `Approve Loan`: Manager is given a list of loans needs to be approved, only the “rich” ones will get approved. (We still do not have GUI for this).

### Customer Side

* `Transfer Money`: Allow customers to transfer money between their own accounts. Also add this part to transaction record.

* `Stock Trade`: Fixed NullPointerException. Allows the rich customers to set up a security account and buy/sell stocks. Also support view profit(realized/unrealized), view opening stocks.

## Other functionality showed during demo

### Manager Side

* `View Customer Transactions`: Allows the manager to view the daily transaction record.

* `Check on Customer`: It allows the manager to check the account information of any customer he/she has.

* `Loan interest auto check`: Although it is not shown in the GUI, we have a function to check date, and interest will increase each month.

### Customer Side

* `Open Account`: Each customer is allowed to have as many accounts as they want. They will be charged a fee when opening account. The first savings account created will become main savings account that is linked to security account to judge whether they qualify for stock trade. Only qualified customer will see the open security account option.

* `Close Account`: Customers could close accounts. Close security account is different, customers are asked to sell all opening stocks before closing security account.

* `Check account information`: Display all information for all accounts of the customer.

* `Check transaction history`: Daily transaction history.

* `Apply for loans / Check loans/ Pay back loans`: Customers could apply for loans and the money would be transferred into their account of choice. They could pay the loans partially, and could check information (apply date, unpaid loan, total value). We also add methods that support an “approve loans” process.

* `Apply for loans / Check loans/ Pay back loans`: Customers could apply for loans and the money would be transferred into their account of choice. They could pay the loans partially, and could check information (apply date, unpaid loan, total value). We also add methods that support an “approve loans” process.

* `Currency Settings`: We have three different currencies, the default one is USD. All the values are recorded in USD, and would be converted to display if needed.

### Supporting functions

* `Error Checking`: We implemented a pop-up window class in GUI, whenever the user makes invalid moves, we call the pop-up window, clean up all the input text boxes, and will not proceed to next page until valid.

* `Data Saving`: We tried to implement a JDBC connection to mysql database to record all the data, but fail to change the configurations to make it work. You could look at the data_driver class to have a brief impression. We plan to update information each time a customer/manager log in/log off and open/close the app.

* `Test Case`: Right now the database is not working, so we implemented several test data.

# Explanation of Classes

Following is a list of classes that were used and an explanation of why they were created and what functions they serve.

## OOP Classes

* `Account`: This class is Represents the superclass of all the accounts. It's attibutes include the money stored in the balance, account number, and type of account. It's functions include setBalance() which makes the balance a specific amount, withdraw() which takes away money from the account and deposit() which adds money to the account balance. The sub accounts were created to identify the different types and hold data for those specific accounts.

  * `Checking`: This subclass represents the Checking account. It's type is "Checking".

  * `Saving`: This subclass represents the Savings account. It's type is "Savings".

  * `Security`: This subclass represents the Security account. It's type is "Security".

* `BankAction`: The superclass for bank activities taken by the cutomer. This was useful to give all the subclasses the date when they were done rather than repeating the goal. The date in which the loan is taken out. It's functions include getYear(), getDay(), and getMonth() which retrieve the year, day or month of when the loan was taken out.

  * `Loan`: This class represents loans by giving it the attributes, money which represents the amount that still needs to get paid to the bank, interest which represents the interest added on to the loan every month The functions increaseLoan() and decreaseLoan() increase the loan to be paid every month, and decrease when a payment is made. monthPass() returns true if a full month passed from the last loan increase.

  * `Transactions`: Transactions extends BankAction. In addition to the date, there is a name for every transaction. The function is getName(). This was created to be able to organize all the transactions by specific dates such as transfers between accounts, withdraws, deposits, and loans.

* `Stock`: This class represent the stocks in the stock market with the attributes of name of the stock and money for the current price of the stock.

  * `OpeningStock`: An instance of an opening stock. It has a value of a share and this class is used to buy, trade, and sell stocks. The functions include getShare(), setShare(int newShare), buyStock(double current_price, int share), and sellStock(double current_price, int share). Created to change the prices of the stocks when bought and sold by the rich customers.

* `Customer`: This class represents the customers or the users of the bank, these attributes include the name of the customer, transactions which is an ArrayList of actions done to the accounts, accounts which is an ArrayList of the customer's accounts, realized profit, unrealized profit, loan, cur for type of currency, the boolean rich which indicates whether the customer is classified as a rich customer in the eyes of the bank manager, mainSaving as a boolean to represent that the customer has a main saving account used to transfer money to the security account and hasSecurity boolean to check if the Customer has a security account yet. We chose to make the boolean rich rather than a rich customer subclass because the only difference was the

permission to access these accounts. The functions include, makeSavingsAccount() which adds a savings account to the accounts list, makeCheckingAccount() which adds a checking account to the accounts list, makeSecurityAccount() which adds a security account to the accounts list which only appears in the front end for rich customers, closeAccount() removes an account from the list, applyLoan() which makes a new loan, isRich() returns true is customer is rich which means they have a savings account greater than 5000, exchangeRate() changes the balance to match which currency the customer wants to use, buyStock() buys a stock and sellStock() sells a stock. interestPay() is used to increase the saving account

* `Manager`: This class represents the bank manager with the attribute of money in the balance. The functions include accountFee() which adds money to the balance every time an account is opened or closed and increaseLoans() loops through all the customers and increases the loans if the if a month has passed when from when they last paid or applied. The manager balance also increases when customers pay their loans and make withdrawals. setStockMarket() gives the manager the ability to update the stocks available, increaseLoans() loops through all the customers increasing the loans that have passed a month, and approveLoans() has the manager loop through all the loan reqests made from the Customers and approve the ones from the rich ones.

* `StockMarket`: Represents the stock market by using an ArrayList of stocks, it also allows the manager to set the price, add or remove stocks, and set stock availability. The functions include addStock(Stock newStock), removeStock(Stock stock), setStockAvailability(int accountIndex, boolean avail), setStockPrice(int accountIndex,double value), and getPriceByName(String name).

## GUI Classes

* `AccountInfoScreen`: A frame that shows the customer all the details about their various accounts

* `AccountManagementScreen`: A frame that allows customers to open a new account, close a current account, view current account information, or view current month transaction history.

* `CheckOnCustomerSelection`: A frame that allows the managers to select a customer to look up account information on.

* `CloseAccConfirmScreen`: A frame that makes a customer confirm they want to close a specified account and alerts them of the fee.

* `CloseAccSuccessScreen`: A frame that alerts the customer they successfully closed an account.

* `CloseAccountScreen`: A frame that allows the customer to choose which screen they would like to close.

* `CurrencySettingsScreen`: A frame that allows the customer to decide what currency they would like money displayed in.

* `CustomerWelcomeScreen`: A frame that welcomes customers to the app and allows them to go to the transactions page, go to the account management page, go to the loan options page, go to the trade stocks and security page, change their currency settings, or log out.

* `InvalidEntryPopUp`: A frame that pops up displays an error message and tells the customer to try again. Also clears the textfield.

* `LoanAccountScreen`: A frame that allows customers to choose which account they would like the loan deposited into.

* `LoanActionScreen` : A frame that displays customer account information and whether they are making a loan payment or taking out a loan.

* `LoanScreen`: A frame that allows customers to apply for a loan, pay a loan, or check the status of a current loan.

* `LoanStatusScreen`: A frame that updates the customer on the current status of their loan.

* `LoanSuccessScreen`: A frame that alerts the customer they have successfully either taken out a loan or made a loan payment.

* `ManagerCustomerDisplayScreen`: A frame that shows the manager all of the customers information.

* `ManagerTransactionsScreen`: A frame that allows the manager to see all of their customers transactions from the past month.

* `ManagerWelcomeScreen`: A frame that allows the manager to check customers transactions or check up on a customer

* `MonthlyTransactionsSuccessScreen`: A frame that shows a customer all the transactions from the past month.

* `MyATM`: Entry point of the program opens WelcomeScreen

* `NewAccountConfirmationScreen`: A frame that makes a customer confirm they want to open a specified account and alerts them of the fee.

* `NewAccountSuccessScreen`: A frame that alerts the customers their account creation was successful

* `OpenNewAccountScreen`: A frame that allows customers to decide what type of account they would like to open.

* `SecurityAccountConfirmationScreen`: A frame that alerts the customers their security account creation was successful

* `StockMarketScreen`: A frame that displays all the stocks to the customers.

* `StockOption`: A frame that allows customer to buy or sell stocks.

* `StockSuccessScreen`: A frame that alerts the customer their stock transaction was a success and asks them if they would like to continue using the app.

* `StockWelcomeScreen`: A frame that allows a customer to indicate they want to trade, view their account profit, view their open position, view their trade history, manage their security account, or close their security account

* `TransactionsAccountChoiceScreen`: A frame that allows the user to decide which account they would like to see the transactions for.

* `TransactionsMoneyScreen`: A frame that asks the customer how much they would like to either withdraw or deposit.

* `TransactionsScreen`: A frame that asks the customer what type of transaction they want to do.

* `TransactionsSuccessScreen`: A frame that alerts the customer their transaction was a success.

* `ViewOpenPositionScreen`: A frame that displays all of the stocks a customer has.

* `ViewProfit`: A frame that allows the customer to see their profit information.

* `WelcomeScreen`: A frame that asks the user if they are a customer or a manager.

## Authors - Group Number 2

* Anthony Vargas U58026559

* McKenzie Joyce U41215872

* Qing Zhao U79705507