**Distributed Project**

December 2nd 2018

Name:\*\*\*\*\*\*\*\*

Distributed ATM and Bank System

**ATM – Bank Network Description:**

A series of banks can be started up on various machines. These banks contain a list of accounts which are owned by that bank. ATM’s are run, and connect only to their corresponding bank (each bank can have multiple ATMs, each ATM connects to only one bank). Users can use any ATM to access any account that is owned by that bank as well as of any bank on the inter-bank network.

**ATM Program Description:**

The ATM program starts by initializing a ATM display object that contains all the graphics and display windows. The main ATM program then creates a new socket connection to the Bank server socket. Once connected, the program runs through a state machine which controls which screen the user is displayed. User input and output through the ATM window is fed to the state machine to trigger changes in state, and actions such as sending Requests, and displaying account information.

**Bank Program Description:**

The bank program starts off by reading a file in which all the known banks IPs and ports are listed. A file of accounts is then read, and added to a HashMap that stores each account by an account number. Next the bank program starts two threads which hold open two socket servers, one which will accept connections from other banks and another which will accept connections from ATMs. The bank then keeps trying to connect to all the unconnected banks until all the banks are connected to. Each connection is stored into an object which has a send function. These objects are placed in a HashMap, and accessible by their corresponding bank ID numbers. Once all the other banks are connected to, the bank program allows the user to enter the names of new accounts as well as save the current list of accounts to file.

In the meantime, the socket server threads wait for new connections, once a connection is made a new thread is spawned to handle that connection and the objects passed through it. HashMaps were used to allow for quick look up of accounts and connections, as well as quick insertion of new accounts and socket connections.

**Message Passing:**

Two serialized objects are used during message passing. The ATM always starts by sending a request, and the bank replies with an outcome. A requestHandler retrieves the account number belonging to the account from the Request, looks up the corresponding account, and tries to perform the action. The result of that action is summarized as an Outcome object that is sent back to the ATM from the bank. If an ATM makes a request for an account not from this bank, the bank acts as a relay, and forwards on the request to the bank at which the account is stored. This bank then handles the Requests and sends the outcome back to the ATM through the bank acting as a relay.

**Request:** The request object is created by the ATM. There are 4 types of requests summarized in the table below

|  |  |  |
| --- | --- | --- |
| **Request type** | **Information Passed** | **Description** |
| Log on | Pin (int) Account # (Long) | For validating the account and enabling the user to access the ATM menu for the account |
| Check Balance | Account # (Long) | Return the balance of the account |
| Withdraw | Amount (Double) Account # (Long) | For subtracting from the balance in account |
| Deposit | Amount (Double) Account # (Long) | For adding to the balance of the account |

**Outcome:**

Outcomes are generated by the bank that owns the account. An Outcome can be either true or false and contains a string. It is true when the requested action is complete, and false when it cannot be completed. The string contains the message that is to be displayed to the user, for example a success message for a successful deposit, or the balance on the account from a balance request.

**Concurrency:**

The system is fully concurrent, and allows for multiple ATMs and banks to run at the same time. The bank can handle connections from multiple machines at the same time. Each connection has a dedicated thread that is spawned when the connection is made. By storing accounts in a ConcurrentHashMap in shared memory, thread safe access to Accounts is guaranteed in an efficient way.

**Potential Improvement and Future Work:**

* The ATM display could be made fancier, with custom bank logos.
* Support for sub accounts in multiple currencies.
* Automatic reconnection, if one bank is shutdown other banks should attempt to reconnect when it comes back online.
* An ATM fee for withdrawing/depositing from an account not owned by this ATMs Bank.
* Encryption of the objects being passed through the sockets.
* Placing holds on deposits, and allowing withdrawals in only certain denominations.

**Instructions:** Follow the instructions in README.txt to run the system and demo.