SOFE 4790: Distributed Systems – Final Project Report

Distributed Banking System - ATM

Name: Neil Ramdath

Student ID: 100519195

Submitted to:

Ying Zhu

Due Date:

Dec. 4th 2016

Table of Contents

Abstract

This distributed banking system will allow clients to make simple banking transactions to their own account. This shall include operations such as being able to withdraw from one's own account, deposit into one's own account, and view one's current bank account statement. This will correspond to each client's unique bank account number as well as their name. The server shall act as the bank system, and handle the operations required to process the requests from the client. The client shall act as an equivalent as an ATM for the user, allowing the bank account holder to choose what they would like to do with their account, as well view their account's information.

This report intends to provide the reader with a detailed overview of the distributed banking system. This simple ATM system will give clients the ability to mimic the key functionalities that an ATM provides, whilst implementing Java's RMI service – allowing for remote invocation capability. This report will consist of a project introduction, a detailed requirements specification, and a design analysis consisting of various diagrams. It shall also include a description of the method of implementation of the system, as well as several examples of the systems GUI. The report shall then be summarized in the conclusion.

Project Introduction

A simplistic distributed banking system that shall allow a client to perform basic bank transactions such as being able to withdraw, deposit and view bank account balances. This shall implement the client-server model, which essentially will delegate each class with set of corresponding operations, and create distinction and functionality between the client class (equivalent a bank ATM's UI) and the server class (the bank's back-end processing and client request handler).

The distributed banking system aims to mimic the experience of a bank account holder and an ATM machine. The client shall initiate the RPC by sending a request message to the bank server to execute a specified procedure (ie. deposit, withdraw, etc.) with parameters. The bank server shall then receive the request, execute the specified procedure, and return the results back to the client. This will also involve the use of Java's RMI package which shall allow for remotely-invocated method execution.

Requirements Specification & Design

List of Stakeholders

The individuals identified below are a list of stakeholders with regards to the Distributed Banking System project. This consists of the main users of the system (an ATM) and those who would benefit and utilize it's functionality the most, whether directly or indirectly.

- Primary Stakeholders:
  - The clients making the transaction
- Secondary Stakeholders:
  - The banks the transactions go to
  - Representatives of other banks
  - Bank managers & regulators
- Tertiary Stakeholders:
  - Database administrators
  - Security managers
  - IT/Software Maintenance Engineers

Requirements

- The system should be able to allow users to enter their credentials prior to viewing their account information.
- The system should enable users to withdraw, deposit, and query their account balance.
- The system should be able to dynamically update each user's account information (balance) as they withdraw/deposit money into their account.
- The system should allow the user to perform more than one transaction per session, if they chose to.
- The system should exit when the user specifies so.
- The system should display the current & correct balance when the user chooses to view their account balance.
- If a client selects withdrawal transaction, the system shall prompt the customer to enter the amount to be dispensed.
- If a client is trying to withdraw and has insufficient funds to do so, then the system should prompt the user with an "insufficient funds" statement.

Diagrams

Figure 1: *System Flow of Information Diagram*

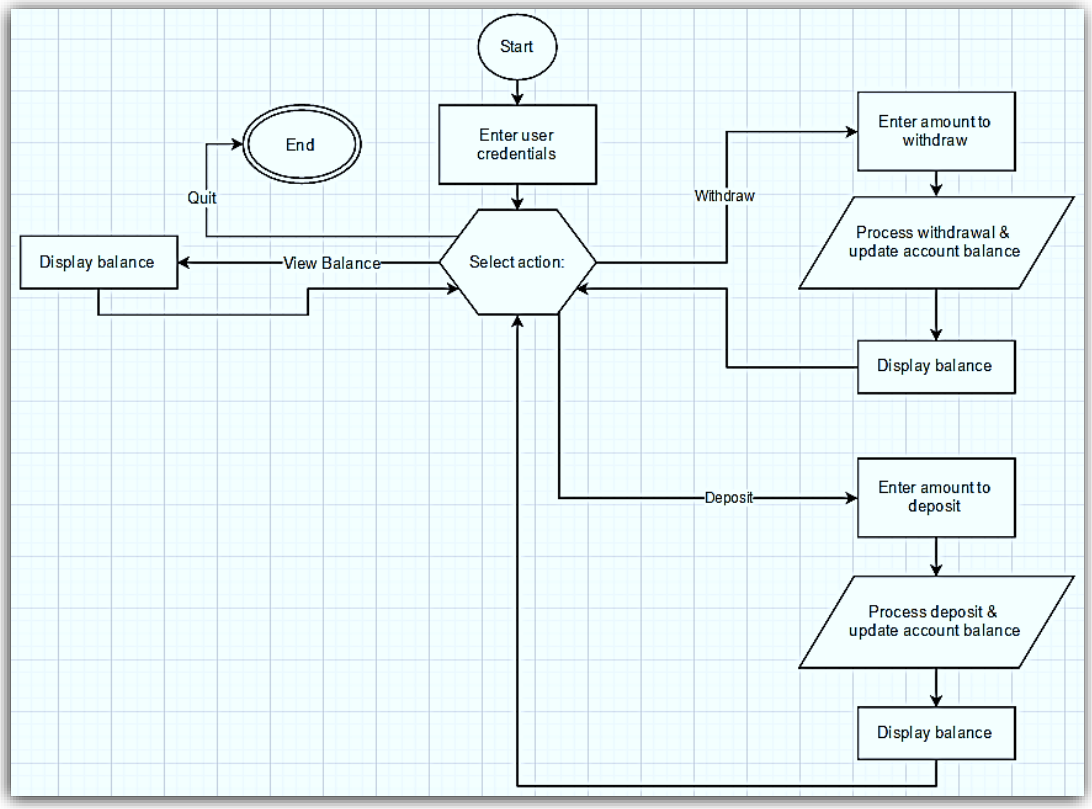


Figure 2: *Class Diagram*

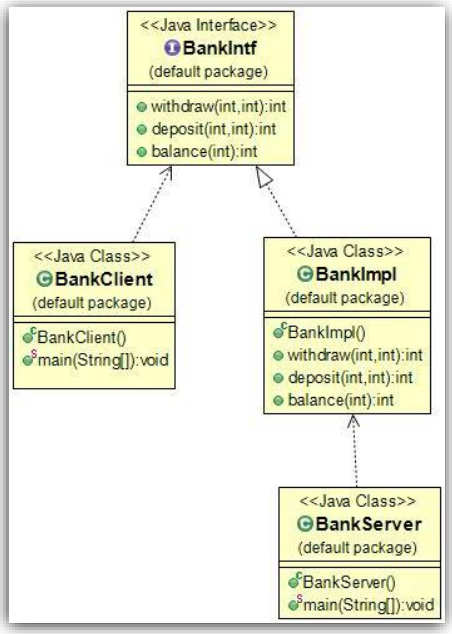


Figure 3: *Main BankClient Class – Sequence Diagram*

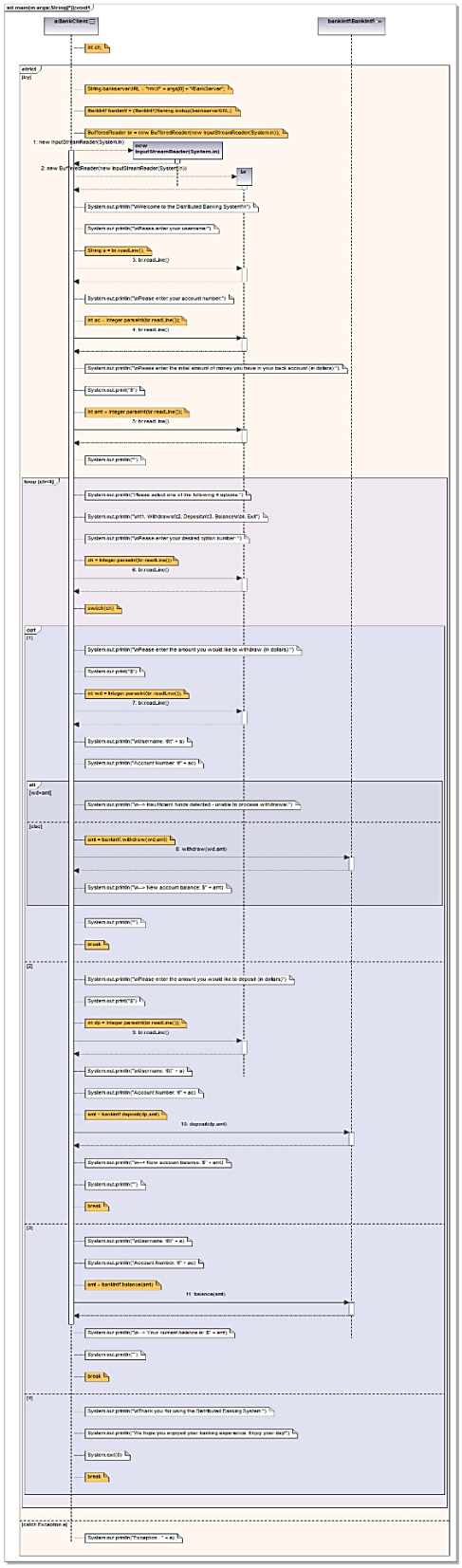


Figure 4: *withdraw Method – Sequence Diagram*

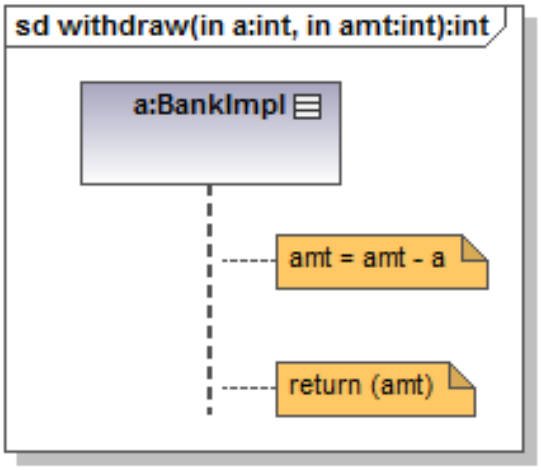


Figure 5: *balance Method – Sequence Diagram*

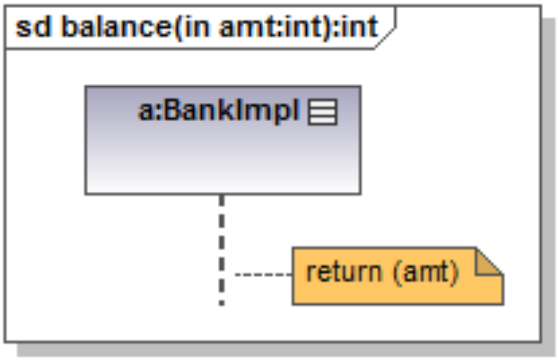
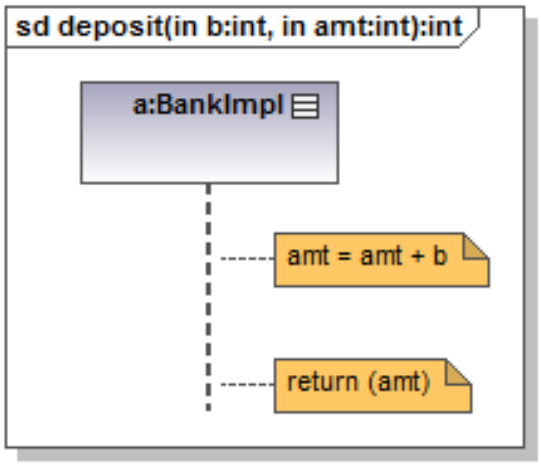


Figure 6: *deposit Method – Sequence Diagram*



Implementation

The implementation of the Distributed Banking System shall closely abide to the predefined preliminary architecture / flow of information. The solution shall utilize the architecture of a distributed system to successfully mimic the functionality of an ATM machine. The distributed banking system will be implemented via Java, as well as it's RMI package. It includes a server class, which shall start and wait on a client to connect to it. It will also include a bank implementation class, which shall define the various methods (transactions/operations) as well as how the server handles those operations. Also, it will include a bank interface class, which shall encapsulate the operations of the various transaction methods available (simply by having parameters passed to it). It will also include a bank client class, which shall act as a user interface for the client to input & display information to.

The BankClient class has been implemented in a manner that handles the user's input and choice selection, and acts as the overall front-end of the system. It also serves the role as the client with regards to the client-server system model, and connects to the active BankServer class. The class serves as an interface for the system (via the command line) and guides the user through the use of the banking system. It greets the user to the system and receives the user's name, account number, and initial amount of money in their account. It then displays a list of options for the user to choose from (ie. whether they would like to withdraw, deposit, view their balance, or exit) and prompts users for the further required information based on their transaction selection. Each of specified transactions correspond to different data processing tasks, each of which is handled and defined by the program's bank-end methods. The BankClient simply passes parameters to the transaction methods for the BankImpl class to handle, which then returns the processed result to the BankClient class and displays the updated value.

The BankImpl essentially defines all of the methods utilized in the Distributed Banking System. This part of the program handles all of the processing required to process transactions such as withdraw, deposit, and view balance. It receives the input given by the user in the BankClient class, processes the data, and updates the account balance respectively. These methods are also defined in the BankIntf class, which serves as a collection of abstract methods which is inherited by the class.

The BankServer class is as the name suggests; it acts as a server for the client to connect to and interact with. It's application is even further extended with the integration of Java's RMI package, which allows for remotely invoked methods and actions between the server and the client.

Screenshots of System Layout

Figure 7: *Withdrawal Transaction*

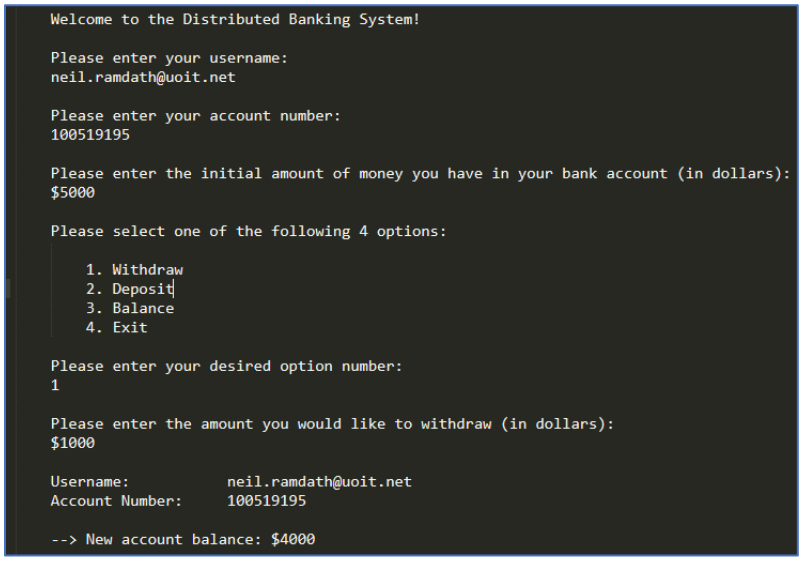


Figure 7: *Deposit Transaction*

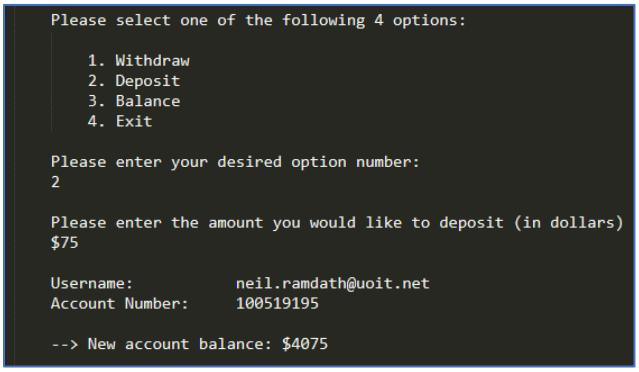
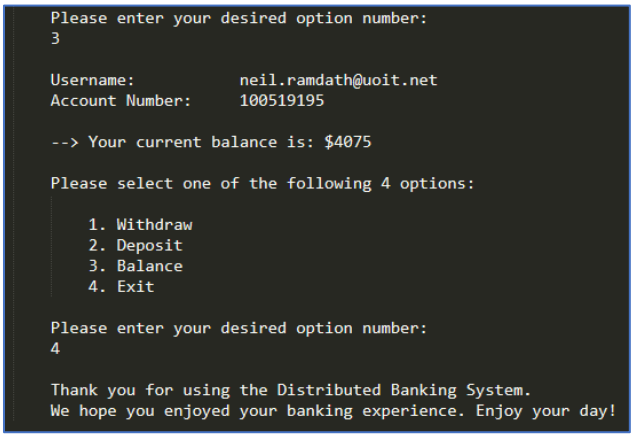


Figure 8: *Balance Inquiry & System Exit*



Conclusion

This distributed banking system aims to mimic the functionality of an ATM machine. The system successfully accepts withdrawals, deposits, as well as balance inquiries from users that have entered their credentials. The system is kept rather simple for ease of use for users, and allows users to simply navigate through the UI of the ATM. The system successfully accepts valid transactions and dynamically updates the user's account balance, encapsulating the bank-end calculations of the banking system itself, allowing it to be easy and efficient to use for quick transactions and account information.