Arizona State University
CSE 434 SP Group 89

Design Document

Sean Berrios

Spring 2023

Table of Contents

1 Description of Message Format

The way I set up my Customer Servers was to create a user application that could be started after the initial commands from the command line were set to set up the Banks Information. In order to run the program you will need to use the following format when running either the customer or bank applications.

java ‘Customer/Bank’ ‘Bank IP’ ‘Bank Port’

Once this command is run you can input the following commands using stdin.
NOTE: Commands issued in the bank will be process from the commands sent from the customer. Those are handled within the code so only the commands listed here need to be specified to the user.

1.1 Required Commands

1.1.1 Open Command

The open command is as required it takes in a command formatted as follows

open ‘customer’ ‘balance’ ‘customer ip’ ‘bank port’ ‘customer port’

The parameters are then parsed as a string and sent to the bank as a message. The bank will take this message and parse it and add a customer via a HashMap which is the Data Structure I have chosen to store all Customer Info. If the Customer infor is stored correctly then the Bank sents a ‘SUCCESS’ message back to the customer and the customer will go back to waiting for a command.

1.1.2 New-Cohort Command

The new-cohort command is implemented as required and takes in a command that is formatted as follows:

new-cohort ‘customer name’ ‘size of cohort’

The new cohort command will parse the message and issue a new cohort message to the bank. When the bank receives the command it will create a new cohort with the issuing customer and a random number of customer for the remaining number of customers available up to the specified size.

Once the cohort is created by the Bank the bank will send the Cohort data Structure as an arraylist. The arraylist is generated from the cohorts data structure inside the bank which is a hashmap that has an arraylist of a cohorts as the value.

Once the customer receives this list it will parse the list and create a new temporary list which does not contain itself and will send the array list to the other customers that are included in the cohort.

1.1.3 Delete-Cohort Command

The delete cohort command is implemented as required and takes in a command formatted as follows:

delete-cohort ‘customer name’

The delete-cohort command parses the message and issues the delete-cohort command to the Bank. When the bank receives this command It will first issue a notification for each customer within the cohort so that they can each delete their cohort database. Once every customer has issued a ‘SUCCESS’ notification to the Bank then the bank will delete the cohort from its database.

1.1.4 Exit command

The exit command is implemented as required and takes a command formatted as follows:

exit ‘Customer name’

The exit command is parsed by the customer and sent to the bank. When the Bank receives the command it will delete the customer from its database and return a ‘SUCCESS’ message back to the customer. When the customer receives this response it will then close the application.

1.1.5 Deposit Command

Deposits a specified amount into the customers account and also updates the customers local cohort for their own balance. Formatted as follows:

deposit ‘amount’

1.1.6 Withdrawl Command

Withdrawls a specified amount from the customers account and updates their local balance for their cohort. Formatted as follows:

withdrawal ‘amount’

1.1.7 Transfer Command

Transfers a specified amount to a specific customer with a specified label ID that can be used for tracking.

transfer ‘amount’ ‘tocustomer’ ‘label’

1.1.8 Lost Transfer Command

The lost transfer command simulates a transfer as lost by listening for a transfer and updating that transfer with information from the sender. The Customer does not send the updated information to any other member in the cohort so when a checkpoint command is issued it is simulated as lost. Formatted as follows

lost-transfer

1.1.9 Checkpoint Command

The checkpoint command once issues waits for a message from each customer in the cohort as an array list. The arraylist contains the current cohort for each customer, I chose to use an arraylist specifically for this purpose as it makes it easier to check if each customer has the same cohort information. If each customer has the same cohort it will print to the console that the checkpoint was successful and information can now be sent to the bank. Otherwise it will print that the checkpoint was unsuccessful and a rollback should be issued by each member. Formatted as follows:

checkpoint

1.1.10 Rollback Command

The rollback command can be issued by only one customer as a time and recieves an arraylist from the bank of the Global cohort info it then deep copies this information into its cohort database to ensure that it has the most recent cohort information. Formatted as follows

rollback

1.2 Commands added for Testing/Integration

1.2.1 Close-Bank Command

This command is a non require command and is used for testing or for closing the bank from a customer process. The command is formatted as follows:

close-bank

This command must be issued from a customer, once issued it will close the Bank server.

1.2.2 Listen-for-cohort

This command is added so that cohorts who are not issuing a new-cohort command can receive messages from their peers for their cohort information. The command is formatted as follows:

listen-for-cohort

The command is issued by all non new-cohort issuing customers. Since the cohort selection is random all customers will need to use this command to receive their cohort information.

1.2.3 Listen

The listen command is used for customers who are awaiting information from the bank for a delete-cohort message. This command is formatted as follows:

listen

This command will listen for the bank to issue a delete-cohort command which the customer will then process and give the bank the result of the operation.

1.2.4 Print-cohort

This command is used for testing and is formatted as follows:

print-cohort
This command can be issued by a customer to view the contents of its current cohort information. It is used for testing to verify whether a cohort was actually deleted or not.

1.2.5 Test

This command is used for testing and is formatted as follows:

Test

This command is issued by a customer and is used for the purpose of printing out all of the data structure in the bank to verify whether a customer/cohort was added/deleted.

1.2.6 Listen For Transfer

The listen for transfer command listens for a transfer, unlike the lost transfer command this command sends the updated cohort information to the rest of the customers in the cohort. This

way when a checkpoint command is issued each customer will have the same cohort information and a checkpoint will have been successful. Formatted as follows:

listen-for-transfer

1.2.7 Create-dummy

The create dummy command was a command I made to make testing a little easier. It creates a dummy cohort for a customer so you can issue an open command on any customer and essentially communicate over the dummy cohort. Formatted as follows:

create-dummy

1.2.8 Recieve New Cohort

The receive new cohort command will be issued by the non transferring parties in a cohort and will receive a new cohort form the customer that is currently processing the listen for transfer command. Formatted as follows:

receive-new-cohort

1.2.9 Send Checkpoint

The send checkpoint command will be used while the checkpoint command is being issued by one customer. The checkpoint command will listen for a specific IP number, the customer with that IP will send a send checkpoint command to the requesting customer with their cohort information.

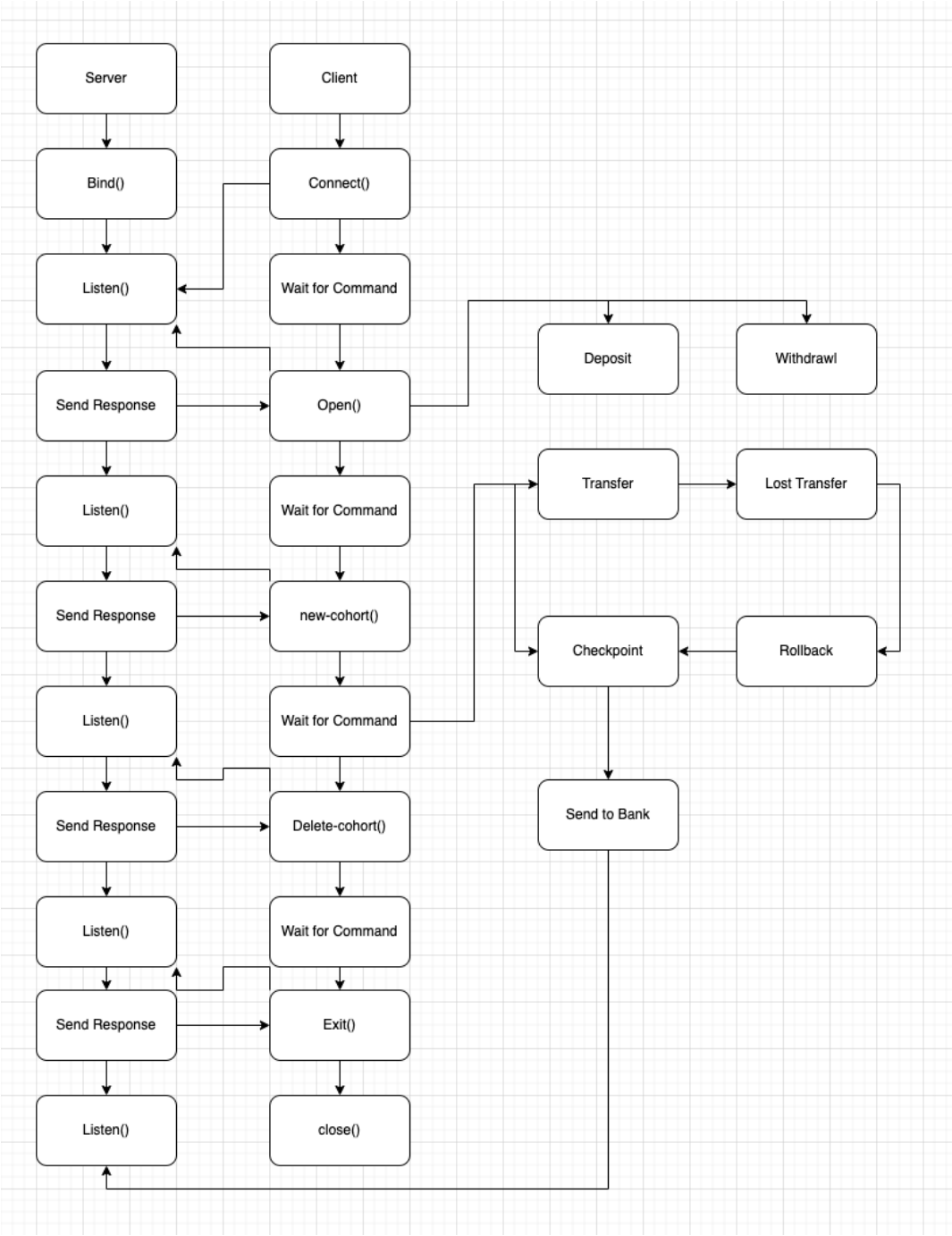
send-checkpoint ‘requestor-ip’ ‘requestor-port’

1.2.10 Send to Bank

The send to bank command should only be used once a successful checkpoint has been received. This command can be issued by any customer at that point and will update the cohort info for the bank to the most recent checkpoint achieved by the customers in the cohort. Formatted as follows

send-to-bank

2 Time-Space Diagram



3 Data Structures and algorithms/ Design Implementations

3.1 Data Structures Used For Bank

3.1.1 Customer Data Structure

The customer Data Structure I used for the Bank Is a HashMap Data Structure. The Hashmap contains a key as the name of the customer and value as another Hashmap that contains all of the customers information.

I chose this format because it makes it easier to be able to move data around and format data as I need for all the different functionality needed for the assignment. Having a hashmap with a hashmap was helpful because I could access all of the data I needed with only the customers name.

3.1.2 Cohort Data Structure

The Cohort Data Structure I used was also a hashmap but for this map I used a different implementation than the customers data structure. For this structure I still used the customers name, but this time it is only the issuing customer that is used as the key of the cohort structure. And then for the value I have an Arraylist the contains a hashmap of all of the list members information.

I chose this format because it still allows me to use hashmap functionality for the bank. Then when I need to access a specific cohort I could look through all of the customers listed and use another method to initiate an Arraylist that I can send as a packet to a customer.

3.2 Data Structure Used For Customer

3.2.1 Cohorts Data Structure

For the customer data structures I only used an Arraylist of hashmaps to store the customers cohort information. I chose this implementation because it is how I designed the cohorts in the bank. It also allows for a more compact list of customer information that I could loop through and be able to break off sections when needed to remove a customer so that the customer can send messages to the rest of the cohort without needing to include itself.

3.2.2 Channel Class

While I did add a channel class and used a Hash Map Data Structure in my code to store these channels. I ultimately decided to go in a different direction with issuing a rollback by just using the bank as the central location for a rollback. I left these Data Structures in the Customer Code but they ultimately are not being used. I initially created them to create a data base of channels that could be tracked with the label identifier so that rollbacks could decide which database to roll back to. This would still be useful if in the future multiple transfers were being made and the bank was not updated after the transfer the customers could rollback only to that last transfer that was

actually lost. This way only the minimum amount of transfers would be wiped. To save time I decided to omit this as it wasn't specifically required in the project document.

3.3 Helper Methods

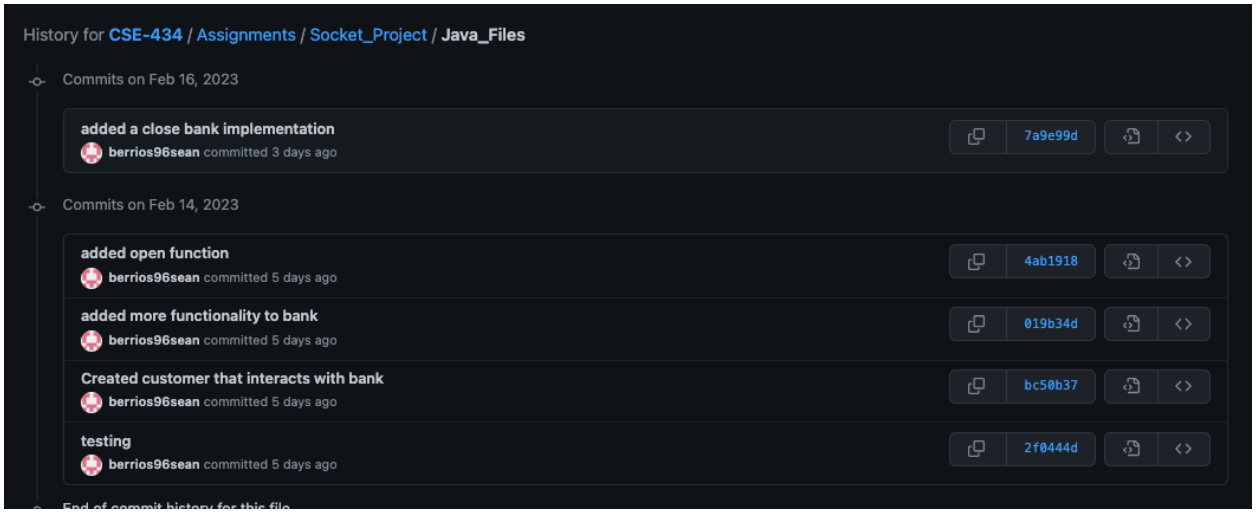
I used a large variety of helper methods which are documented within my code and would be to large to include here.

I have included some JavaDoc comments so that these methods can be viewed more easily.

4 Screenshot of Commits

4.1 Migration From Local to general.asu.edu

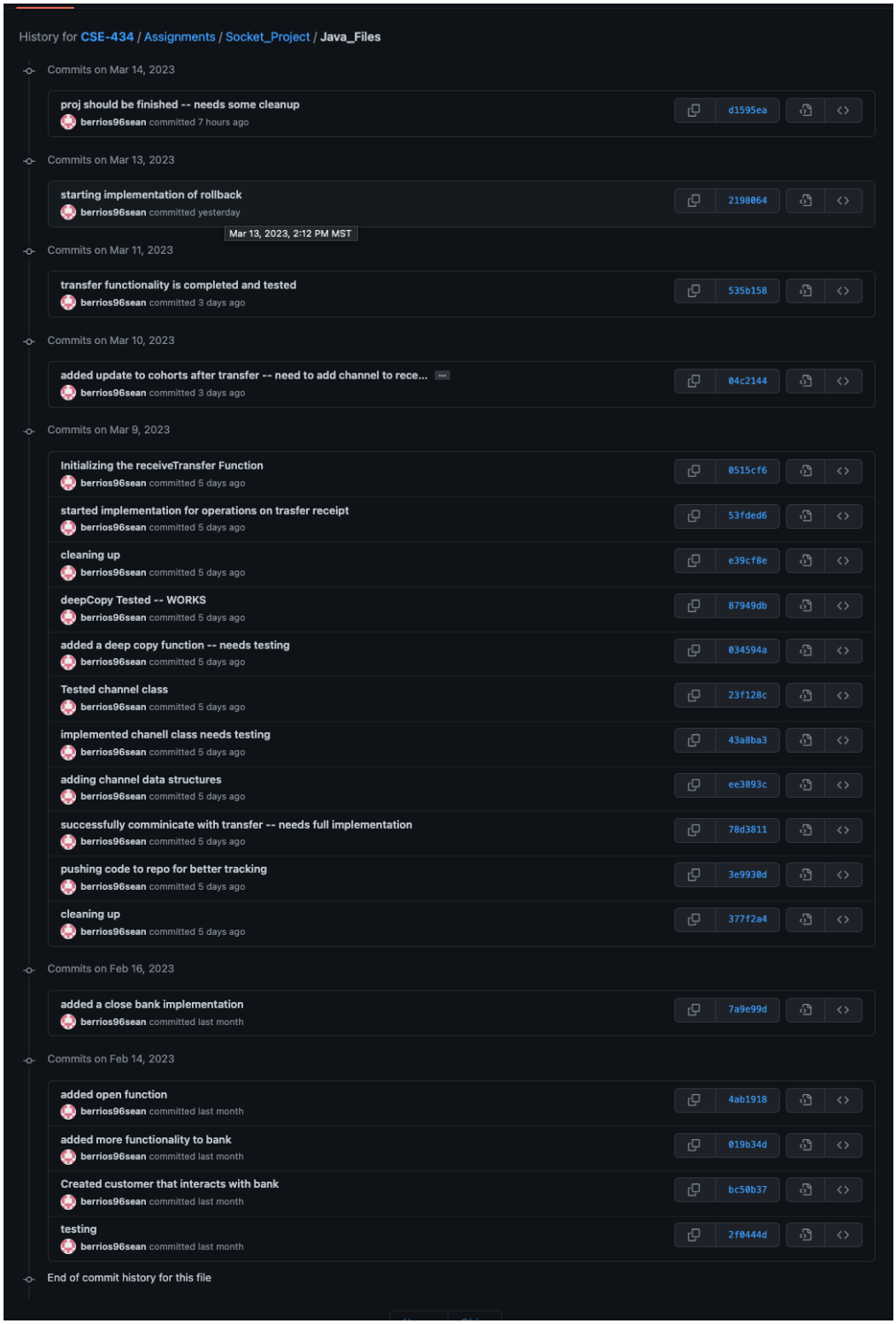
All commits in screenshot are prior to my migration from local to the general.asu.edu server. As a result only the initial connections I was able to make on my local server are included in the commits.
Due to having to need a token to clone my repository on the general.asu.edu server I chose to not use my github repository for the implementation on the general server.



4.2 Migration from general.asu.edu

After the Milestone project I decided It would be helpful form me to migrate my code back to my local PC every now and then to maintain a relatively up to date version control in the event I ran into some issues later on I could review some older commits and track those changes. I did keep this pretty up to date relatively often and did end up

forgetting every now and then but I think my commit history this time accurately reflects changes I have made during the implementation of the full project.



5 Link to Video Demo

The link to the youtube video was still processing at time of submission. The shareable link that youtube provided is what is included. I listed this video as public on my personal channel which I will also provide a link for in the event that the shareable link is not accessible.

Personal Channel Link: https://www.youtube.com/@seanberrios7827/featured

Demonstration Video Link: https://youtu.be/FPwbjz2o0sU

5.1 Video Timestamps

Timestamps are also included on the youtube video

Timestamps
0:00 Intro
0:50 Compiling Programs
1:10 Start Bank/ Open Customers
3:30 create new cohort
5:10 deposits/withdrawals 1
5:55 Transfer 1
7:15 Checkpoint 1
9:20 deposits/withdrawals 2
10:20 Transfer 2
11:10 Checkpoint 2
13:25 deposits/withdrawals 3
14:05 lost transfer
14:50 checkpoint 3
15:25 rollback
16:20 deposits/withdrawals 4
16:50 lost transfer 2
17:25 checkpoint 4
18:00 rollback 2
19:50 delete cohorts
21:05 exit
21:34 close bank