CISC327 Assignment 4

Backend Design Document

Loaft Inc.

Tayyab Ahmad - 10197212
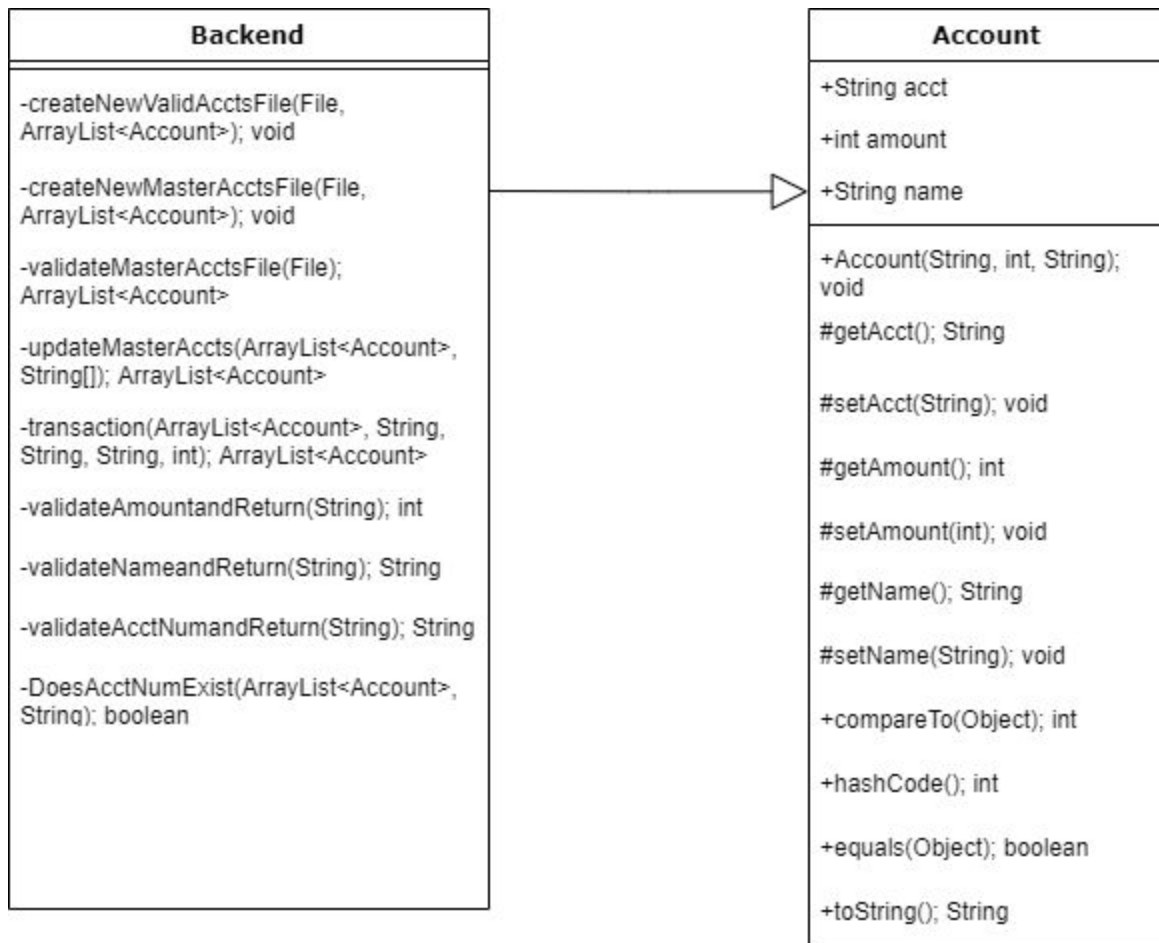Luvit Chumber -10190467
Samir Mekhdi - 10191295
Jessica Wong - 10181646

Solution Diagram
Please refer to solution diagram in Appendix A for flow of solution.

Class Diagram
The backend class is used as the main entry point for the QUINTERAC backend. It manages all file and transaction handling, along with log and error outputs. The Account class is used to store master accounts as objects. It also implements Comparable<Object> to allow sorting of objects.

Backend
-createNewValidAcctsFile(File, ArrayList<Account>); void
-createNewMasterAcctsFile(File, ArrayList<Account>); void
-validateMasterAcctsFile(File); ArrayList<Account>
-updateMasterAccts(ArrayList<Account>, String[]); ArrayList<Account>
-transaction(ArrayList<Account>, String, String, String, int); ArrayList<Account>
-validateAmountandReturn(String); int
-validateNameandReturn(String); String
-validateAcctNumandReturn(String); String
-DoesAcctNumExist(ArrayList<Account>, String); boolean

Account
+String acct
+int amount
+String name
+Account(String, int, String); void
#getAcct(); String
#setAcct(String); void
#getAmount(); int
#setAmount(int); void
#getName(); String
#setName(String); void
+compareTo(Object); int
+hashCode(); int
+equals(Object); boolean
+toString(); String

backend:
The backend file does a complete check of inputs before writing and updating the files: masteraccts.txt and mergedtsf.txt. First, it will validate the master accounts file by performing checks on the old file using `validateMasterAcctsFile()`. It will then parse the information and save it as an *Account* object. After a successful validation, the master accounts file is updated to either add an account, delete an account or transfer money to/from an account. The `transaction()` method does a check according to the transaction requested from the TSF file. For example, if *'NEW'* is read, the program will check to see if the account number exists. The program will then sort the

information on the updated master accounts file and overwrite the previous file with the updated information. Using the same updated information, it will update the previous valid accounts file with just the users account number.

`Account:`

The account program houses an object composed of the account number, the amount in the users account and the name of the user. There are various methods to set and get the values for each of the objects contents. The last couple of methods in this file are override methods. compareTo is used to sort collections using the account numbers. When doing a `compareTo`, `hashCode` will be used to to override. `equals` overrides to implement proper method when checking if ArrayList<Account>.contains(acct). The final method, `toString`, is used to return a string that combines account number, amount and name.

Appendix A

validateAcctNameandReturn

if (in == null || !in.matches("^[\\sa-zA-Z0-9]*$")) —True→ isAlphanumeric = false;

False ↓

if (in.length()>=3 && in.length()<=30 && isAlphanumeric) —False→ name = "NotValid"

True ↓

return name ←

if (acct.equals("NotValid")) —True→ System.err.println ("Error with account number")

False ↓

if (amount == -1) —True / Text→ System.err.println ("Error with amount")

False ↓

if (name.equals("NotValid")) —True→ System.err.println ("Error with name")

False ↓

Account cache = new Account(acct,validateAmountandReturn(amountStr),name);
masterAccts.add(cache);

↓

return masterAccts

↓

masterAccts = updateMasterAccts(masterAccts,mergedTsfFileNames);

↓

masterAccts = transaction(masterAccts,type,acctFrom,name,amount);

transaction

transaction

Account temp = new Account(acct,amount,name);
int idx,currentAmount;

case "NEW" case "NEW" case "NEW" case "NEW" case "NEW"

masterAccts.add(temp) || masterAccts.remove(temp) ||
masterAccts.set(temp)

return masterAccts

Comparator<Account> c = Collections.reverseOrder();
Collections.sort(masterAccts,c);

createNewMasterAcctsFile(masterAcctsFile,masterAccts);

createNewMasterAcctsFile

Files.write(masterAcctsFile.toPath(),output.getBytes());

createNewValidAcctsFile(validAcctsFile,masterAccts);

createNewValidAcctsFile

Files.write(masterAcctsFile.toPath(),output.getBytes());