CISC 327 BACKEND OFFICE UNIT TESTING

CLOUD PROGRAMMING

Ly Sung 10150760
Tyler Mizuyabu 10141746
Calvin Che 10158050

Test inputs (transactions):

MasterAccounts.txt	MergedTransactions.txt
10150012 12300 Lucy	WD 00000000 10150012 12300 Lucy
	WD 00000000 10150013 12300 Lucy
	WD 00000000 10150012 123000 Lucy
	CR 10150012 00000000 000 Lucy
	CR 10150011 00000000 000 Tyler1
	CR 10150013 00000000 000 Tyler2

All accounts from the master accounts file will be stored in an arraylist called accounts.

Each transaction in the merged transaction summary file will be read and turned into a String array where index 0 is the type of transaction to be made. For withdraw, index 2 and index 3 will be passed in addInt() as parameters for account number and amount to withdraw respectively. As for creating the account, index 1 and 4 which represent the account number and account name will be given to createAccount().

addInt() and createAccount() will check if the given transaction line is valid or invalid by checking with the old master accounts file.

Decision Coverage:

```
148         public static int getAccount(String accountNum) {
149                 for (int i = 0; i < accounts.size(); i++) {
150                         if (accounts.get(i).equals(accountNum)) {
151                                 return i;
152                         }
153                 }
154                 return -1;
155         }//end getAccount
```

```
81  public static void addInt(String accountNum, int amount) {
82      int index = getAccount(accountNum);
83      if (index != -1) {
84          int current = Integer.parseInt(accounts.get(index)[1]);
85          current += amount;
86          //current amount needs to be greater than 0
87          if (current < 0) {
88              System.out.println("Failed constraint: account balance is negative");
89          } else {
90              String temp = Integer.toString(current);
91              while (temp.length() < 3) {
92                  temp = "0".concat(temp);
93              }
94              accounts.get(index)[1] = temp;
95          }
96      }
97  }//end addInt
```

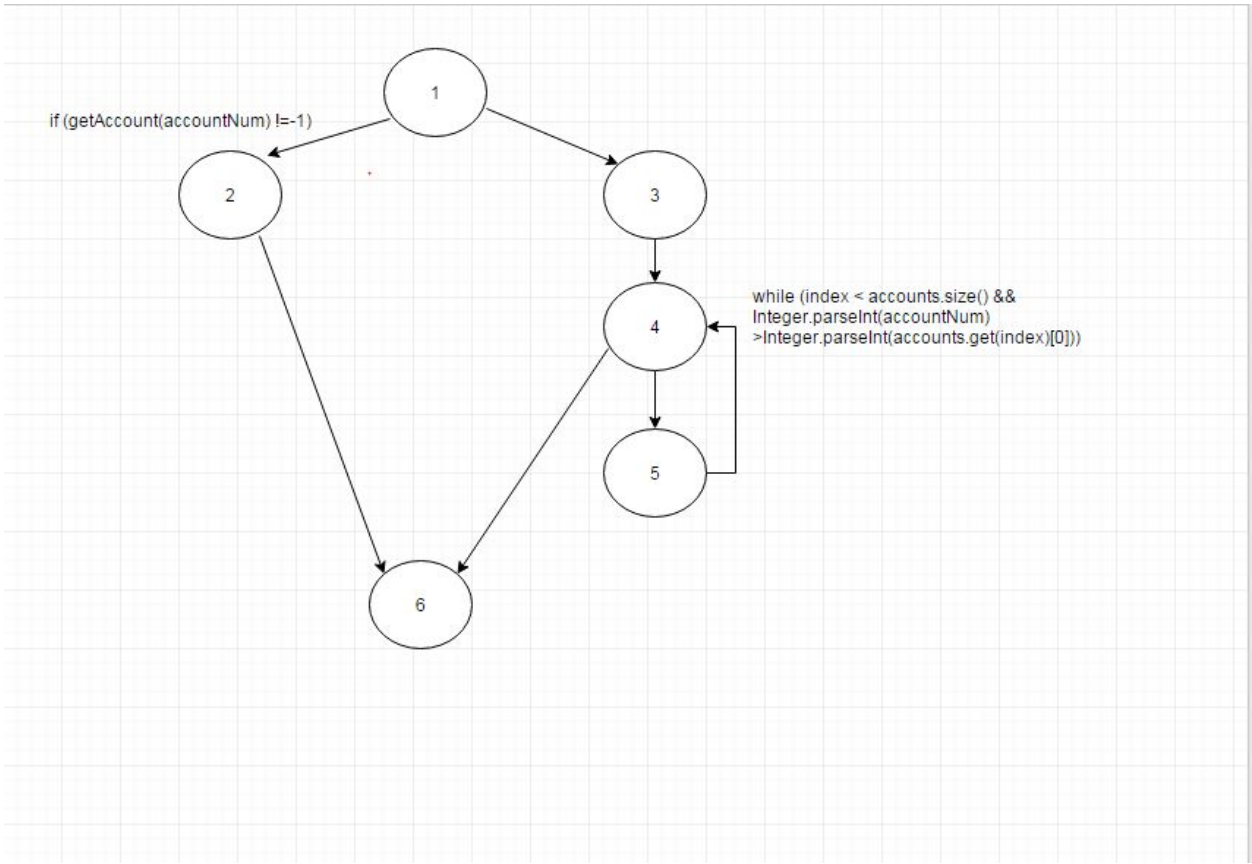
arrayList ***accounts*** stores all the accounts read from the master accounts file.
TransactionLine reads each line from the merged transaction summary file as a string array.

Decision	**accountNum input**	**Amount (to withdraw) input**	**Test**	**Output**
getAccount Method				
1:true	accountNum = 10150012 arrayList accounts = {[“10150012”, “12300”, “Lucy”]}	0	T1	i -> the index of where the account number is found in the arraylist
1:false	accountNum = 10150013 arrayList accounts = {[“10150012”, “12300”, “Lucy”]}	0	T2	-1 -> indicating that the account number is not in the arraylist
addInt Method				
2:true	arrayList accounts = {[“10150012”, “12300”, “Lucy”]} TransactionLine = [“WD”, “00000000”, “10150012”, “12300”, “Lucy”]	Amount = 12300	T3	“Valid Account”

	accountNum = 10150012			
2:false	arrayList accounts = {[“10150012”, “12300”, “Lucy”]} TransactionLine = [“WD”, “00000000”, “10150013”, “12300”, “Lucy”] accountNum = 10150013	Amount = 12300	T4	“Invalid Account”
3:true	arrayList accounts = {[“10150012”, “12300”, “Lucy”]} TransactionLine = [“WD”, “00000000”, “10150012”, “123000”, “Lucy”] accountNum = 10150012	Amount = 123000	T5	“Valid Account Failed constraint: account balance is negative”
3:false	arrayList accounts = {[“10150012”, “12300”, “Lucy”]} TransactionLine = [“WD”, “00000000”, “10150012”, “12300”, “Lucy”] accountNum = 10150012	Amount = 12300	T3	“Valid amount” “Change in balance: -12300”

Path Coverage:

```
104         */
105         public static void createAccount(String accountNum, String accountName) {
106                 if (getAccount(accountNum) != -1) {
107                         System.out.println("Failed constraint: account number already exists");
108                 } else {
109                         int index = 0;
110                         //finds the right place to store the new account
111                         while (index < accounts.size() && Integer.parseInt(accountNum) > Integer.parseInt(accounts.get(index)[0])) {
112                                 index++;
113                         }
114                         accounts.add(index, new String[] { accountNum, "000", accountName });
115                 }
116         }//end createAccount
117
```



Paths	**Nodes**	**accountNum Input**	a**ccountName Input**	**Output**
P1	1,2,6	accountNum = 10150012 arrayList accounts = {[“10150012”, “12300”, “Lucy”]}	“Lucy”	arrayList accounts = {[“10150012”, “12300”, “Lucy”]}
P2	1,3,4,6	accountNum = 10150011 arrayList accounts = {[“10150012”, “12300”, “Lucy”]}	“Tyler1”	arrayList accounts = {[“10150011”,”000”,”Tyler1”],[“10150012”, “12300”, “Lucy”]}
P3	1,3,4,5,4,6	accountNum = 10150013 arrayList accounts = {[“10150012”, “12300”, “Lucy”]}	“Tyler2”	arrayList accounts = {[“10150012”, “12300”, “Lucy”],[“10150013”,”000”,”Tyler2”]}

Failure Report

After running the tests for the first time It was found that all tests resulted in failures except for T3 in the branch coverage of withdraws functionality. As well the path coverage of creating an account failed in producing correct results. The reason being that in our code we were accidently comparing the string representation of a string array to the account number passed into our getAccount() method. As such this always produced an index of -1 no matter what account number was being searched for, and since this method is used by all other methods in backend to check whether an account exists it caused our tests to fail. The solution was instead of doing accounts.get(i).equals(...) in BackEnd.java, we did accounts.get(i)[0].equals(...). Now properly comparing account numbers.

Tests were run from a jar file, **Tests.jar**. The code used to run the tests can be found in its respective java file. The process it uses in running the tests is first it does the branch coverage tests for the addInt() and getAccount() method used in the withdrawal process. When running the branch coverage tests for addInt(), the expected result is console output so the test code helps differentiate the console output with a header describing what test case is being run and

lines separating the results of the test cases. When testing the getAccount() method, the output is the return value printed out in formatted text in the testlog.txt. This is also the case for the path coverage tests. In general testlog.txt shows the expected output for every test and if applicable will have the resulting output of each test (cases where it won't show is cases where the results are console output and must be checked by hand).

Also would like to note that print statements were added to the addInt() function for comparing test results.