Account Creation Transactions Method – Statement Coverage Testing

Backend.py : Lines 44-48

```
                def create_account(accountNumber, accountName):
1                       global transactions, accounts_filename, accounts


2                       accounts[accountNumber] = ("0", accountName)
3                       return
```

Statement	Input	Test	Executed
1	Test1.txt	T1	Yes
2	Test1.txt	T1	Yes
3	Test1.txt	T1	Yes

Test1.txt:

NEW 1234567 000 0000000 test

EOS 0000000 000 0000000 ***

Statement Coverage Testing:

Because every statement in the program to be executed at least once, giving us confidence that every statement is at least capable of executing correctly.

Failures:

None, test causes all statements of the method to execute.

Withdraw Transaction Method – Path Coverage

```
def withdraw(accountNum, amount):
    global accounts
    try:
        currAcc = accounts[accountNum]
    except:
        print("Account doesn't exist")
        return
    currBalance = int(accounts[accountNum][0])
    if currBalance - amount > 0:
        print("Account overdrawn, transaction not completed")
    else:
        accounts[accountNum] = (str(int(accounts[accountNum][0]) - amount), account[accountNum][1])
    return
```

Test Method – White Box Input Partition Coverage

We are using white box input partition coverage to test the withdraw function to ensure every possible state the merged transaction summary file and the master accounts file can be in is tested with relation to the withdraw function. The types of inputs are generalized to certain situations and states the file can be in, assuming the frontend takes care of any "illegal inputs" the system may have to deal with.

All test files are in the folder "inputTestFiles"

Input 1 – No accounts in the master accounts list file (trying to withdraw before creating an account), but a "WDR" transaction exists in the merged transaction summary file.
Expected result: printed to the console - "Account doesn't exist" and nothing is written to the master accounts file.
See folder "Input1" for test inputs.

Input 2 – 1 unique existing account with withdraw transaction.
Expected result: specified amount is subtracted from the user with the specified account number's existing balance and the new information is written to the master accounts file and the valid accounts file.
See folder "Input2" for test inputs.

Input 3 – 2 separate accounts with withdraw transactions.
Expected result: specified amount is subtracted from the each of the respective users with the specified account number's existing balance and the new information is written to the master accounts file and the valid accounts file.
See folder "Input3" for test inputs.

Input 4 – 1 unique account with multiple (3) withdrawals.
Expected result: Account dictionary is updated upon every revolution of the loop and the end balance that is written to the master account file is reflective of all three transactions.
See folder "Input4" for test inputs.

Input 5 – User withdraws an amount larger than what is currently in their account.
Expected result: printed to the console "Account overdrawn, transaction not completed" and the system moves to the next transaction.
See folder "Input5" for test inputs.

Input 6 – One account in the merged transaction summary file is in valid but another one is valid.
Expected Result: The invalid account is caught and the "Account doesn't exist" error is printed to the console, but the system continues to work according to input type 2 for the valid account.
See folder "Input6" for test inputs.

Input 7 – Master files don't exist.
Expected result: "Can't find master accounts list file" and/or "Can't find merged transactions summary file" written to the console.
No input files, run backend code by itself.

Test Report
Failures:
Originally wasn't accounting for negative account balance, changed to fit the requirements.

Rest of the tests run and pass as expected.