Assignment 5: Back Office Unit Testing

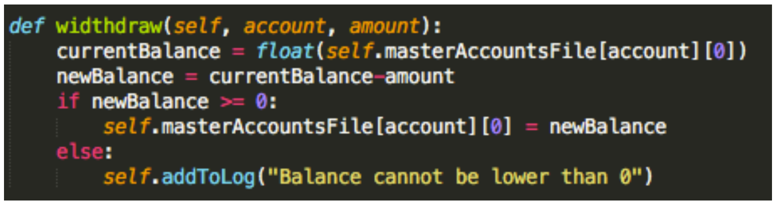
Team Members:

- **Thomas Cartotto**
    - Student Number: 10130262
    - Number of Hours: 4-5
    - Contribution
        - Created expected log files for each test
        - Wrote code that creates a log for each test and then validates it with the expected output.
        - Wrote this report
- **Josh Daiter**
    - Student Number: 10127054
    - Number of Hours: 4-5
    - Contribution
        - Wrote the tested back office functions + method that runs the tests
        - Wrote functions that write New Master account file and New Valid account file.
- **Ryan Fredrickson**
    - Student Number: 10130487
    - Number of Hours: 4-5
    - Contribution
        - Figured out the most effective WBT method for each of the methods we were going to test.
        - Wrote expected inputs and outputs for each of the WBT methods.

Withdraw Transactions

White Box Method: *Statement Coverage*

The first two lines within the function will be executed regardless of the inputs and will achieve 100% code coverage. The third line within the code splits the code into 2 possible paths. To get 100% coverage of the whole function, we need a test case where currentBalance is > amount and one case where currentBalance is <= amount.



```
def widthdraw(self, account, amount):
    currentBalance = float(self.masterAccountsFile[account][0])
    newBalance = currentBalance-amount
    if newBalance >= 0:
        self.masterAccountsFile[account][0] = newBalance
    else:
        self.addToLog("Balance cannot be lower than 0")
```

Tests and their expected I/O:

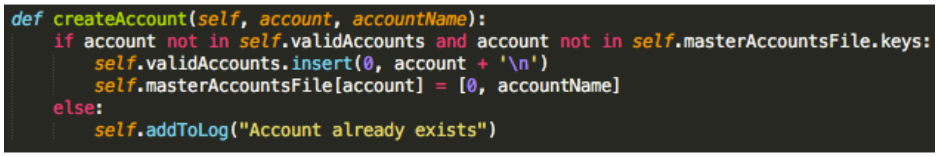
Test Number	Test Input into backend	Expected output (Log File)
Test 1	WDR 4444444 5555 0000000 *** EOS	withdrawalSuccess
Test 2	WDR 4444444 -1000 0000000 *** EOS	withdrawalFail(ballanceLowerThanZero)

Test Results:

Create Account

White Box Method: *Decision Coverage*

To gain 100% code coverage of the following function, the if statement on the first line but be triggered to go both ways, thus covering both decisions that can be made. To get into the first section we must have the account number in both the master and Valid accounts file. If either are false, the second section will be executed. To test all possibilities 4 test cases will be needed as there are two inputs to the decision (00, 01, 10, 11).



```
def createAccount(self, account, accountName):
    if account not in self.validAccounts and account not in self.masterAccountsFile.keys:
        self.validAccounts.insert(0, account + '\n')
        self.masterAccountsFile[account] = [0, accountName]
    else:
        self.addToLog("Account already exists")
```

Tests and their expected I/O:

Test Number	Test Input into backend	Expected output
Test 3-both true	NEW 4545454 000 0000000 Hello EOS	creationSuccess
Test 4-first false	NEW 1111112 000 0000000 Hello NEW 1111112 000 0000000 Hello EOS	creationFailed(accountExists)
Test 5-second false	NEW 1111111 000 0000000 Hello EOS	creationFailed(accountExists)
Test 6-both false	NEW 1111111 000 0000000 Hello NEW 1111111 000 0000000 Hello EOS	creationFailed(accountExists) creationFailed(accountExists)

Test Results: