Prestige Worldwide: Quinterac
Assignment 5

Alex Beaumont Stidwill	(10176777)
Devin Alldrit	(20013756)
Javier Sanchez Mejorada	(20024581)
Logan Roth	(10176909)

Account Creation Testing

Statement Coverage Testing

Test Cases

The table below shows the test case used for the account creation testing.

Test Name	Test Case	Input	Expected Output
CreateAcctT1	Statement coverage Lines 1-4 in transaction.__createAcct(self, mafDct)	**masterAcctsFile.txt:** 1234567 111 abc **mergedTransactionSummaryFile.txt:** NEW 1234567 000 00000000 abc EOS NEW 7654321 000 00000000 cba EOS	**masterAcctsFile.txt:** 1234567 111 abc 7654321 000 cba **terminal:** line 1 Account 1234567 already in Master Accounts File. A new account cannot use a number already in the File line 2 line 3 line 4

Source Listing

Below is the source code that the account creation testing uses to perform its tests.

```
def __createAcct(self, mafDct):
    """
    Performs the necessary update to the MAF to add an account.

    @param mafDct The dictionary describing the MAF
    """
    # Ensure account is not already in MAF
    if self.acct1 in mafDct.keys(): #1
        print('line 1')
        print('Account {} already in Master Accounts File. A new account '
            'cannot use a number already in the File'.format(self.acct1)) #2
        print('line 2')
    else: #3
        print('line 3')
        mafDct[self.acct1] = {'Name': self.acctName, 'Balance': 0} #4
```

```
        print('line 4')
```

Transaction Inputs

Below are the transaction inputs from the Merged Transaction Summary File to cover each test case.

NEW 1234567 000 00000000 abc
EOS
NEW 7654321 000 00000000 cba
EOS

Failure Table Results

Below is the test report reporting the results of the test runs of the Back Office.

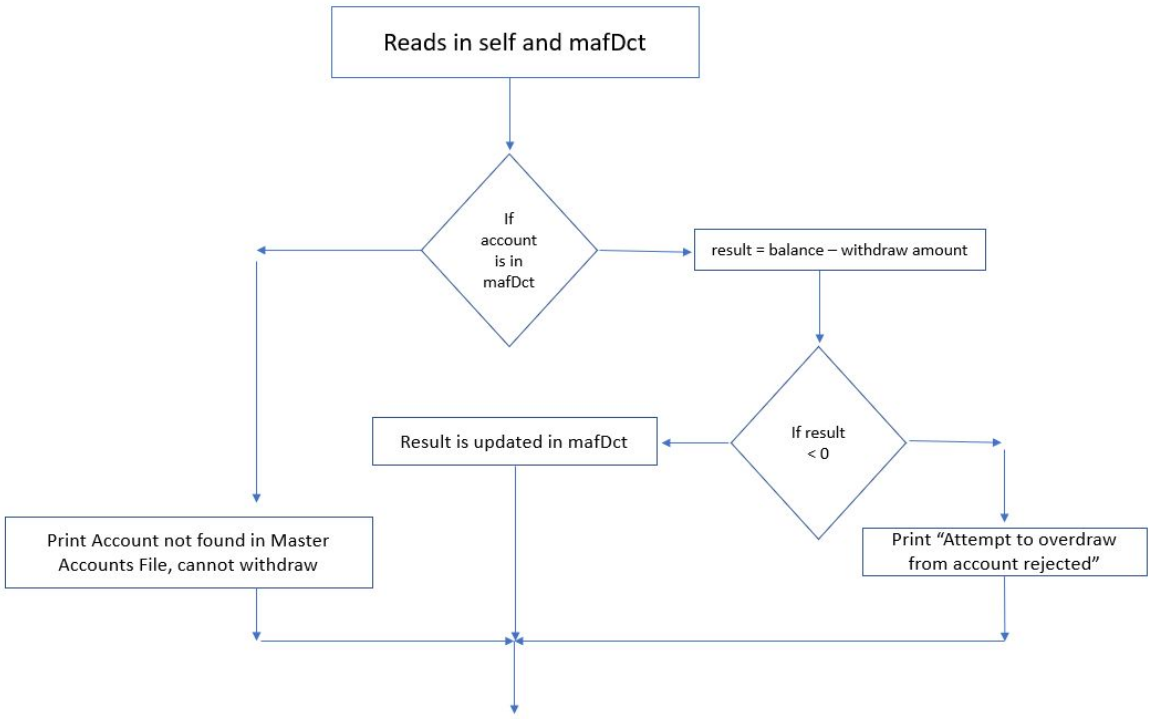
Test Name	**Test Objective**	**Output Error**	**Code Error**	**Fix**
CreateAcctT1	Statement coverage of createAcct method	**terminal:** `Transaction in invalid format EOS`	'EOS' was being checked for in transactions.py	'EOS\n' is what should be checked for

After fixing this error, there were no more observed failures while performing statement coverage on the createAcct method. All 4 lines were reached and the expected output was seen in the terminal and the master accounts file. The test was validated manually due to the simplicity of only running a single test.

Withdraw Testing

Decision Coverage Testing

The diagram below shows the paths of the decision coverage testing.



Test Cases

The table below shows the test case used for the account creation testing.

Test Name	Test Case	Input	Expected Output
WithdrawT1	If the account is in mafDct and the result is a negative	**MasterAccountsFile.txt:** 1234567 000 abc **mergedTransactionSummaryFile.txt:** WDR 1234567 123 00000000 abc EOS	**MasterAccountsFile.txt:** 1234567 000 abc **Terminal:** Attempt to overdraw from 1234567, withdraw rejected.
WithdrawT2	If the account is in mafDct and the result is a positive	**MasterAccountsFile.txt:** 1234567 123 abc **mergedTransactionSummaryFile.txt:** WDR 1234567 123 00000000 abc EOS	**MasterAccountFile.txt:** 1234567 000 abc **Terminal:** Withdraw passed
WithdrawT3	If the account is not in the mafDct	**MasterAccountsFile.txt:** 1234567 123 abc **mergedTransactionSummaryFile.txt:** WDR 9999999 123 00000000 abc EOS	**MasterAccountsFile.txt:** 1234567 123 abc **Terminal:** Account 9999999 not found in Master Accounts File, cannot withdraw

Source Listing

Below is the source code that the account creation testing uses to perform its tests.

```
def __withdraw(self, mafDct):
        """
        Performs the necessary update to the MAF to withdraw from an
account.
        @param mafDct The dictionary describing the MAF
        """
        # Ensure account is in MAF
        if self.acct1 in mafDct.keys():
            result = mafDct[self.acct1]['Balance'] - self.amt
            # Ensure withdraw will result in a valid amount in the account
            if result < 0:
                print('Attempt to overdraw from account {}, withdraw '
                      'rejected.'.format(self.acct1))
            else:
                mafDct[self.acct1]['Balance'] = result
                print('Withdraw Passed')
        else:
            print('Account {} not found in Master Accounts File, cannot '
                  'withdraw.'.format(self.acct1))
```

Transaction Inputs

Below are the transaction inputs from the Merged Transaction Summary File to cover each test case.

WithdrawT1:

WDR 1234567 123 00000000 abc
EOS

WithdrawT2:

WDR 1234567 123 00000000 abc
EOS

WithdrawT3:

WDR 9999999 123 00000000 abc
EOS

Failure Table Results

Through the various tests conducted on the backend, no errors were found for the withdraw method. This is due to careful implementation of code in assignment 4, and an adequate use of the paired programming technique for both assignment 4 and assignment 5.

Team Member Assessment

Alex Beaumont Stidwill

Assignment 4 tasks were mainly split between Logan and I, while assignment 5 tasks were mainly split between Devin and Javier. For assignment 4, I worked on the design document which described the overall structure of our solution as a UML diagram, and included a description of each class. I spent slightly over 3.5 hours working on assignment 4.
For assignment 5, I helped with the selection of the systematic white box unit test method for each section of code that we were testing. Additionally, I formatted the document to ensure that each section was covered and then once the test reports were included in the document, I peer reviewed the assignment. This totalled for roughly one hour spent on assignment 5.

Devin Alldrit

The team split up tasks for assignment 4 and 5 between all the members. Javier and I focused our efforts mainly on assignment 5 doing statement coverage and decision testing on the back office using the pair programming technique. The whole team met initially to breakdown all the work and develop the overall design and plan for the back office and back office testing. This took about 1 hour and then the white box testing took about 3.75 hours leading to a total of 4.75 hours. I also was in charge of tagging the git repository and submitting everything.

Javier Sanchez Mejorada

The team decided to split our team of four into two teams in order to implement proper paired programming techniques. Devin and I were focused on implementing the majority of assignment 5. We gathered as a team to brainstorm the best whitebox testing methods to use and then Devin and I implemented the two testing methods on the createAccount method and the withdraw method. This took myself about 3.75 hours to complete my part of assignment 5. Furthermore, throughout assignment 4 the team through the ideation of the UML diagram and discussing what the backend would look like and what the best implementation would be. This took about 1 hour, totally 4.75 hours for assignment 4 and 5.

Logan Roth

As Alex mentioned above, Assignment 4 was mainly split between him and I, while Assignment 5 was mainly split between Devin and Javier. For assignment 4 I was primarily responsible for coding the back office with the guidance of Alex who created the design documentation and helped with creating the structure for the solution. As a team we all discussed how the backoffice should be implemented in a roughly one hour meeting. I spent roughly 6 hours working on assignment 4 between the actual coding and planning out the structure. For Assignment 5, I helped with the selection of the white box unit test methods for each section of code that we were testing using my low-level knowledge of the code I had written to help with selection of which methods would work best and peer-reviewing the final work. Overall I spent roughly 1 hour working on assignment 5.