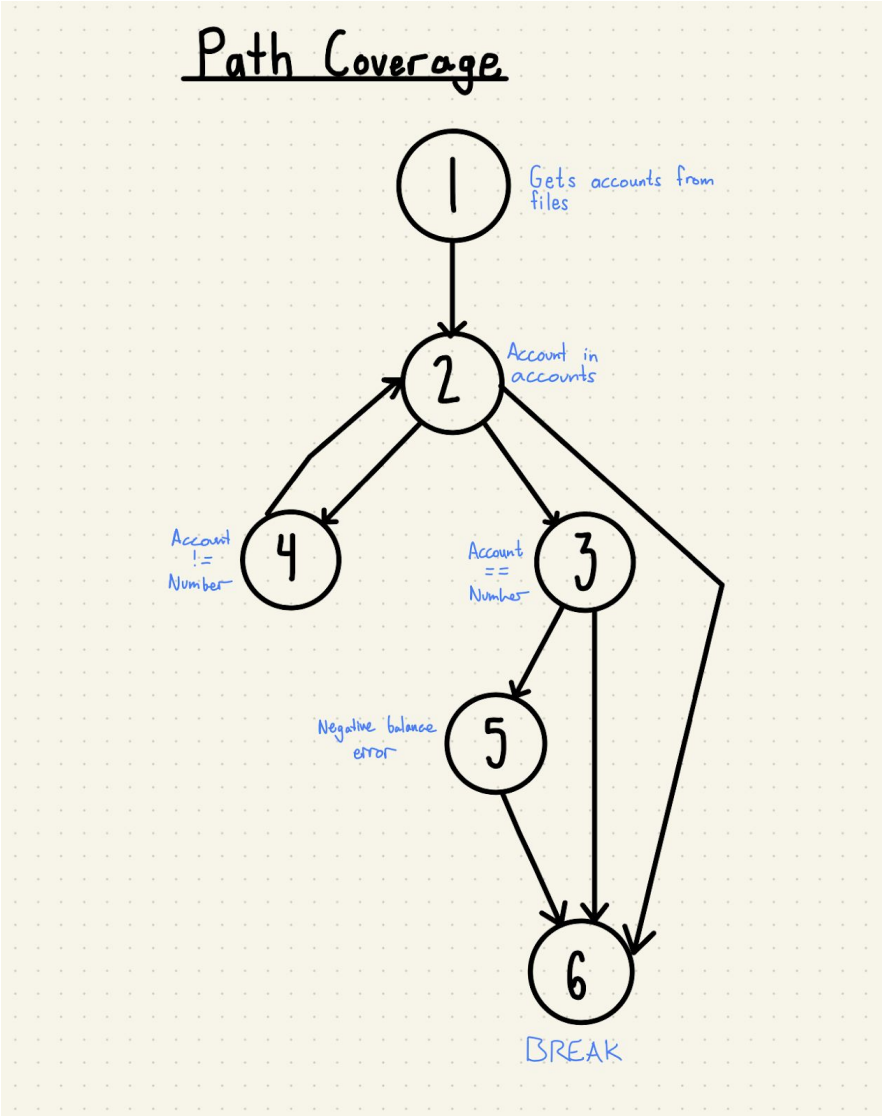
Withdraw Transaction:

Whitebox testing method: Path Coverage.

```
# Start Method
        def withdrawAccount(self, number, amount):
            if len(self.accounts) == 0:     # No Accounts
               logging.error(Error.noAccounts.value)
            for account in self.accounts:
               if account.accountNumber == number:
                  if account.updateBalance(-amount) == Error.negativeBalance:
                        logging.error(Error.negativeBalance.value + "- for account: " +
                        account.accountNumber)
                  Break
# End Method
```



Test Case Analysis

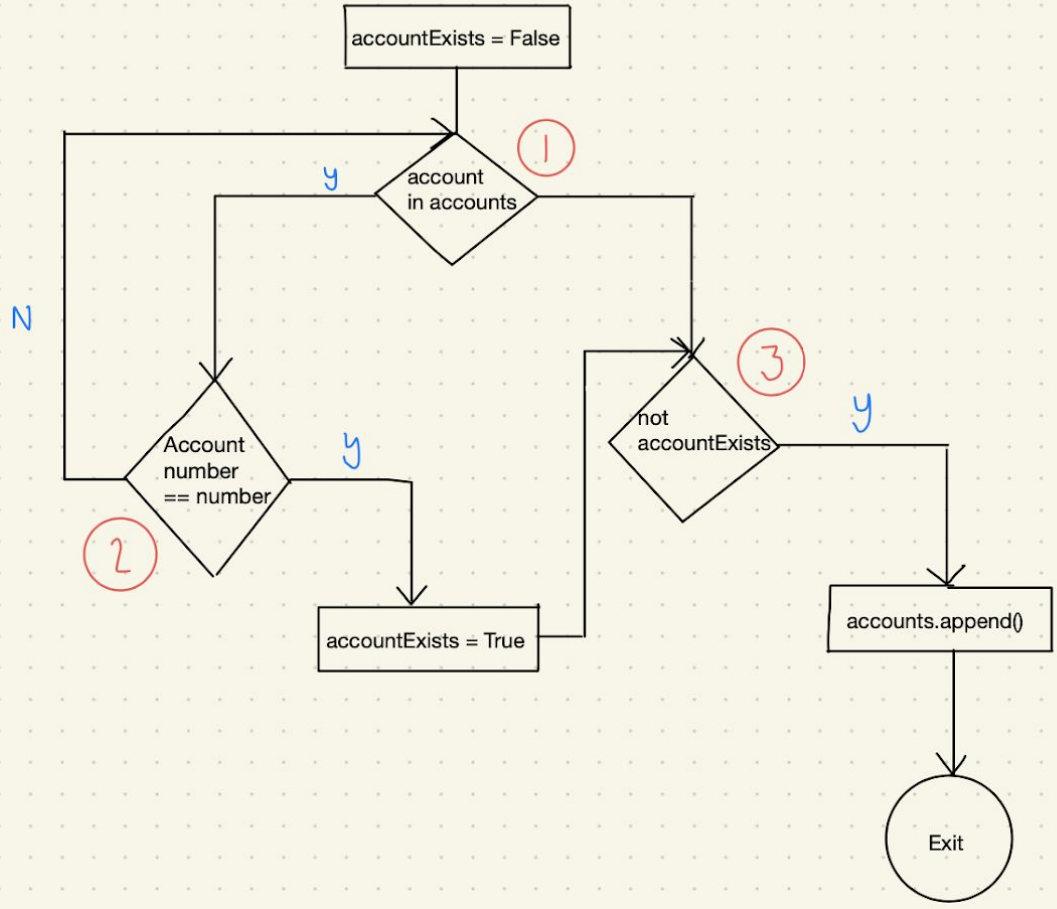
Test Case Name	Purpose	Arguments	Path
testPathOneWithdraw	Test covers path where there are no accounts to withdraw from.	noAccounts - Empty list of Account objects.	1, 2, 6
testPathTwoWithdraw	Test covers path where the account is the first element of the account list and successfully updates the account balance.	Accounts - List of Account objects. Transactions - Transaction input from user.	1, 2, 3, 6
testPathThreeWithdraw	Test covers path where the account is the first element of the account list and withdraw causes a negative balance error.	Accounts - List of Account objects. Transactions - Transaction input from user.	1, 2, 3, 5, 6
testPathFourWithdraw	Test covers path where the accounts list is iterated through and successfully updates the account balance.	Accounts - List of Account objects. Transactions - Transaction input from user.	1, 2, 4, 3, 6
testPathFiveWithdraw	Test covers path where the accounts list is iterated through and withdraw causes a negative balance error.	Accounts - List of Account objects. Transactions - Transaction input from user.	1, 2, 4, 3, 5, 6

Create Account Transactions:

Decision Coverage

```
def createAccount(self, number, name):
    accountExists = False
(1) for account in self.accounts:
(2)     if account.accountNumber == number:
            accountExists = True
            logging.error(Error.accountExists.value + "- for
account: " + account.accountNumber)
            break

(3) if not accountExists:
        self.accounts.append(Account(number, name, True))
```



Analysis

Decision	Accounts Input	Account Number Input	Test
1: True	Accounts exist	Input account number != any account number	T1
1: False	No accounts exist	Input account number != any account number	T3
2: True	Accounts exist	Input account number == any of the account number	T2
2: False	Accounts exist	Input account number != any accounts number	T1
3: True	No accounts exist	Input account number != any accounts number	T3
4: False	Accounts Exist	Input account number == any of the accounts number	T2

Test Case Analysis

Test Case Name	Purpose	Arguments
testDecisionOne	Tests the first decision by testing if the users transaction input of account number doesn't exist.	Accounts - List of Account objects. Transactions - Transaction input from user.
testDecisionTwo	Tests the second decision by testing if the users transaction input of account number already exists.	Accounts - List of Account objects. Transactions - Transaction input from user.
testDecisionThree	Tests the third decision by testing if any accounts exist, thus account number input will always be a valid account number.	noAccounts - Empty list of Account objects.

Test fixtures and user input - In both testing files

Test Fixture Name	Purpose
accounts	This fixture reads from the master accounts file and creates a list of Account objects to be tested against.
noAccounts	This fixture is the same as the accounts fixture, but just reads from an empty master accounts file.
transactions	This fixture reads from the merged transaction summary file (user input) and stores each line in a list. Each line is tested with the corresponding test method.