Computer Science 4472B Assignment 3

Integration Testing

April 7th, 2022

Prof. Kostas Kontogiannis

Group 20

Amanpreet Gill

Ositadinma Arimah

Atulpreet Johar

Bohr Deng

Table of Content

Main Procedure

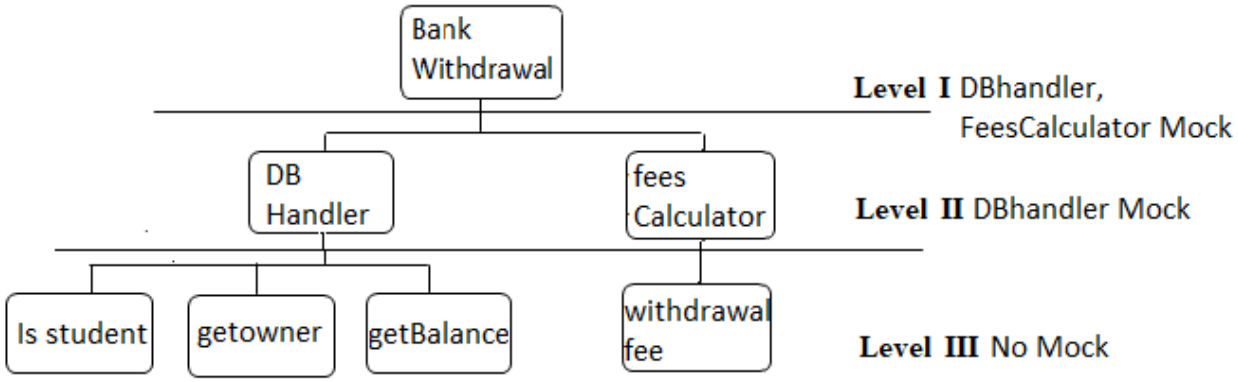
Since the top-down integration strategy focuses on testing the top layer or the controlling subsystem first. The three controlling subsystems we diagnosed and tested were:

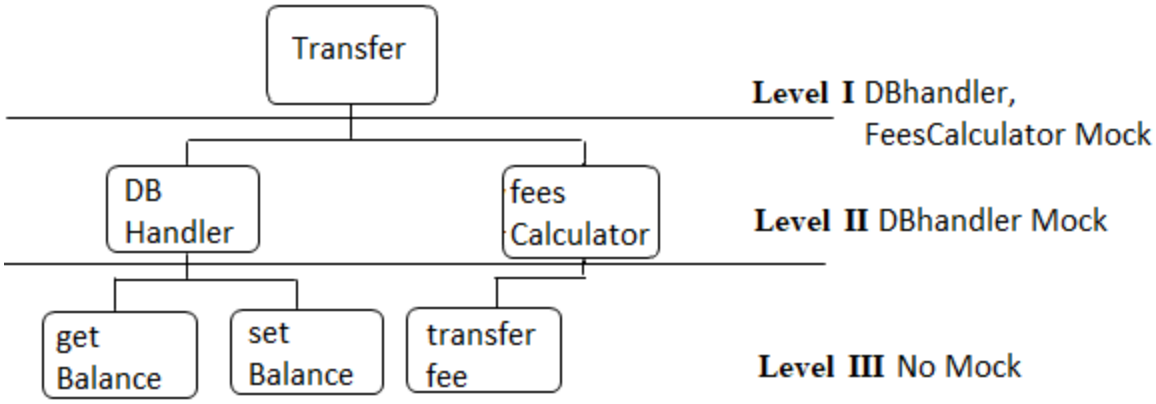
- Withdrawal
    - AllMocksTest - DBhandler and Fees calculator mock
    - DBMockTest - only DBhandler mock, actual fees handled through class
    - NoMockTest - all active classes instead of stubs

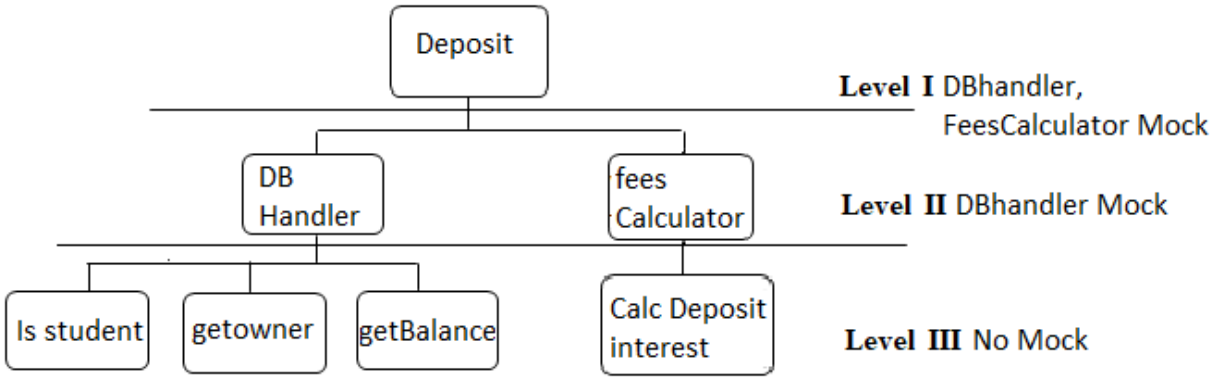
- Transfer
    - BankTransferTest - DBhandler and Fees calculator, as well as DBhandler mocks
    - BankTransferTestNoStubs - all active classes substituted with stubs

- Deposit
    - BankDepositTest - DBhandler and Fees calculator mock, after first stub only DBhandler mock, and lastly no mock setup chronologically

The relationship between the controlling and child components are as follows:




Transfer
Level I DBhandler, FeesCalculator Mock
DB Handler
fees Calculator
Level II DBhandler Mock
get Balance
set Balance
transfer fee
Level III No Mock


Deposit
Level I DBhandler, FeesCalculator Mock
DB Handler
fees Calculator
Level II DBhandler Mock
Is student
getowner
getBalance
Calc Deposit interest
Level III No Mock

Bank Withdrawal

Step 1: Mock DBHandler and FeesCalculator

Test #	amount	balance	isStudent	dOW	fees	expectedBalance	P/F
1	50	1000	true	Sat	0.00	950.00	P
2	50	1000	true	Wed	0.05	949.95	P
3	50	1000	false	Sun	0.05	949.95	P
4	50	999	false	Fri	0.10	948.90	P
5	50	1001	false	Fri	0.05	950.95	P
6	50	10001	false	Fri	0.00	9951	P

Some tests that were dOW sensitive were initially failing because the BankWithdrawal.perform() method considered the dayOfWeek as the current dayOfWeek. This is appropriate for the use case, however for testing it would ignore the passed in day of week parameter.

I created a separate method in BankWithdrawal that would return the current day of the week. This allowed me to spy the object and stub this method with the test parameter. Once I made this change all tests passed.

Step 2: Mock Just DBHandler

Test #	amount	balance	isStudent	dOW	fees	expectedBalance	P/F
1	50	1000	true	Sat	0.00	950.00	P
2	50	1000	true	Wed	0.05	949.95	P
3	50	1000	false	Sun	0.05	949.95	P
4	50	999	false	Fri	0.10	948.90	P
5	50	1001	false	Fri	0.05	950.95	P
6	50	10001	false	Fri	0.00	9951	P

These tests all passed, no new errors were raised.

Step 3: No Mocks, All Real Objects

Test #	amount	balance	dOW	fees	expectedBalance	P/F
1	50	1000	Sat	0.00	950.00	P
2	50	1000	Wed	0.05	949.95	P

The test user used in the test was a student and so non-student test cases could not be run. There was a multiple write error that the DBHandler object was throwing. I suspected this was due to 2 reasons:

- I was redeclaring the DBHandler object for each test
    - I moved this out to a @BeforeAll method, so it was only declared once
- The DBConnection seems to never have been getting closed
    - I added a close connection in the teardown @AfterAll method, although I suspect this might not have caused the error but it is recommended to close the connection manually
- I also tracked the starting balance of the DB to reset the balance after all tests are complete
    - Between each test the account balance was set to the test parameter, balance, to keep consistent tests
    - This could be improved by having a test database so we aren't working on the production database in this example

Bank Transfer

Description

The files BankTransfer.java and BankTransferTestNoStubs.java are classes used to test the functionality of the transfer method.

The classes test the Transfer method in two different ways.

Firstly, the parameterized tests “FeesCalculatorTrasnsferTest_[two/one/none]” are implemented to test whether the correct fee was designated for a transfer transaction.

Secondly, the parameterized tests “TrasnsferBalanceTest_[two/one/none]” are implemented to check the correctness of the two account balances after the transfer transaction.

Step 1: Mock DBHandler and FeesCalculator

1. Create the two parameterized tests with two stubs each (FeesCalculatorTrasnsferTest_two, TrasnsferBalanceTest_two). One mock class for feesCalculator object and one mock class for dbHandler object.

2. Create the two parameterized tests with one stub each (FeesCalculatorTrasnsferTest_one, TrasnsferBalanceTest_one). Replace the feeCalculator stub for the original feesCalculator object and test using a dbHanlder stub.

3. Create the two parameterized tests with all original objects (FeesCalculatorTrasnsferTest_none, TrasnsferBalanceTest_none). Ie, test without the use of stubs.

Test #	amount	from acct balance	to acct balance	student	transfer fee	P/F
1	50	100	100	true	0.01	P
2	50	100	10001	true	0.005	P

Step 2: Mock Just DBHandler

Test #	amount	from acct balance	to acct balance	student	transfer fee	P/F
1	50	100	100	true	0.01	P
2	50	100	10001	true	0.005	P

Step 3: No Mocks, All Real Objects

Test #	amount	from acct balance	to acct balance	student	transfer fee	P/F
1	50	100	100	true	0.01	P
2	50	100	10001	true	0.005	P

Bank Deposit

Step 1: Mock DBHandler and FeesCalculator

Test #	amount	balance	studentstatus	Expected fees	expected balance	P/F
1	101	1001	T	0.01	1001+(101*1.01)	P
2	101	1001	T	0.005	1000+(101*1.005)	P
3	50	5001	T	0.05	5001+(50*1.005)	P
4	50	1000	T	0.0	1000+50	P
5	501	5001	F	0.01	5001+(501*1.01)	P
6	501	10001	F	0.0	1000+(501*1.005)	P
7	100	10001	F	0.005	10001+(100*1.005)	P
8	100	1000	F	0.0	1000+100	P

Step 2: Mock Just DBHandler

Test #	amount	balance	studentstatus	Expected fees	expected balance	P/F
1	101	1001	T	0.01	1001+(101*1.01)	P
2	101	1001	T	0.005	1000+(101*1.005)	P
3	50	5001	T	0.05	5001+(50*1.005)	P
4	50	1000	T	0.0	1000+50	P
5	501	5001	F	0.01	5001+(501*1.01)	P
6	501	10001	F	0.0	1000+(501*1.005)	P
7	100	10001	F	0.005	10001+(100*1.005)	P
8	100	1000	F	0.0	1000+100	P

Step 3: No Mocks, All Real Objects

Test #	amount	balance	studentstatus	Expected fees	expected balance	P/F
1	101	1001	T	0.01	1001+(101*1.01)	P
2	101	1001	T	0.005	1000+(101*1.005)	P
3	50	5001	T	0.05	5001+(50*1.005)	P
4	50	1000	T	0.0	1000+50	P
5	501	5001	F	0.01	5001+(501*1.01)	P
6	501	10001	F	0.0	1000+(501*1.005)	P
7	100	10001	F	0.005	10001+(100*1.005)	P
8	100	1000	F	0.0	1000+100	P