Table report for back-end

These test cases are designed to test the back-end merged daily transaction file. The front end was designed in a way that accepted only good inputs and protected the program from bad input. Due to this design the test cases for back-end are truncated. In order to fully complete this assignment we developed J-Unit test cases for User class (Buy,Sell, and AddCredit) using test methods such as Decision Coverage Testing, Basicblock Testing, Input partitioning white box Testing.

Function name	Test Description	Daily transaction Input	Result
Create	Test creation of new user	01_newPerson_FS_200.0	Passed
Create	Test creation of new user that already exists	01_pibbles_FS_1000.0	Passed
Delete	Test Deletion of user exists and all associated tickets	02_Jamie_BS_20.0	Passed
Delete	Tests Deletion of user that does not exist	02_noexist_AA_200.0	Passed
AddCredit	Test if the updated credit of the user after the AddCredit transaction is correct.	06_buyer_BS_10.0	Passed
AddCredit	Test if the user's username is correct.	06_buyer_BS_10.0	Passed
Buy	Test if buyer’s credit is updated after buy transaction	04_Random event_Michael_buyer_1_50.0	Passed
Buy	Test if the buyer's username is correct.	04_Random test_Michael_buyer_1_50.0	Passed
Buy	Test if the ticket is deleted from tickets file when all tickets are sold out.	04_Random event_Michael_buyer_1_50.0	Passed
Buy	Test if the number of tickets is adjusted correctly after a ticket sell	04_Random event_Michael_buyer_1_50.0	Passed

Sell	Test if the buyer's username is correct.	04_Brazil_seller_4_33.0	Passed
Sell	Test if event for sale is created correctly.	04_Brazil_seller_4_33.0	Passed
Refund	Test if refund amounts correct for seller and buyer	05_testttt_newPerson_200.0	Passed
Refund	Test for refund when seller has insufficient funds	05_buyer_newPerson_800.0	Passed
Refund	Test for refund no existing buyer	05_noexist_newPerson_200.0	Passed
Refund	Test for refund no existing seller	05_newPerson_noexist_300.0	Passed
Refund	Test for no buyer or seller	05_fake_noexist_23.0	Passed

Table report for Sell()-Buy()-AddCredit()

@Test Sell()
Decision Coverage Testing

Green = good input
Red = bad input

	Test	Event Name Input	Sale Price Input	Ticket Quantity Input	Results
First if= true Second if = true Third if= true Fourth if = true	T1	Test_Event	999.99	100	Passed
First if= true Second if = true Third if= true Fourth if = false	T2	Test_Event	999.99	101	Passed
First if= true Second if = true Third if= false	T3	Test_Event	1000	100	Passed
First if= true Second if = true Third if= false	T4	Test_Event	1000	101	Passed
First if= false	T5	Random event(this event already exists)	999.99	100	Passed
First if= false	T6	Random event	999.99	101	Passed
First if= false	T7	Random event	1000	100	Passed
First if= false	T8	Random event	1000	101	Passed

First if = true Second if = false	T9	Test_event(16 characters with spaces)	1000	100	Passed
First if = true Second if = false	T10	Test_event(16 characters with spaces)	1000	101	Passed
First if = true Second if = false	T11	Test_event(16 characters with spaces)	1000	101	Passed
First if = true Second if = false	T12	Test_event(16 characters with spaces)	1000	100	Passed

The test cases are designed based on the condition statements, giving good input and bad input to test all possible branches of the condition statements. All the code inside the Sell () method is tested and all the tests passed successfully.

@Test Buy()
Basic block Testing

Green = good input
Red = bad input

	Test	Event Name Input	Seller username input	Ticket quantity input	Buyer credit input	Results
	T1	Random event	Daniel	1	100	Passed
	T2	Random event	Daniel	1	0	Passed
	T3	Random event	Daniel	6	0	Passed
	T4	Random event	Jacob	6	0	Passed
	T5	Random	Jacob	1	0	Passed

		event				
	T6	Random event	Jacob	6	100	Passed
	T7	Random event	Jacob	1	100	Passed
	T8	Random event	Jacob	6	0	Passed
	T9	Not_found_ event	Daniel	1	100	Passed
	T10	Not_found_ event	Daniel	1	0	Passed
	T11	Not_found_ event	Daniel	6	100	Passed
	T12	Not_found_ event	Frank	1	100	Passed
	T13	Not_found_ event	Daniel	6	0	Passed
	T14	Not_found_ event	Frank	6	100	Passed
	T15	Not_found_ event	Frank	1	0	Passed
	T16	Not_found_ event	Frank	6	0	Passed

This method is longer than the others, hence I decided to use basic block testing. The test cases were designed so every basic block is executed at least once. Inside each block, there is a print statement and all of them were printed after running all test cases.

@Test addCredit()
Input partitioning white box Testing

Green = good input
Red = bad input

Test	Credit_to_add input	Results
T1	10	Passed
T2	0	Passed
T3	50.0	Passed
T4	1001.0	Passed

This method is relatively short. The test cases are designed based on the input partitioning white box testing method. The inputs are divided into 4 partitions testing each partition with the simplest partitioning value. All test cases passed successfully.