CAPITAL UNIVERSITY OF SCIENCE & TECHNOLOGY, ISLAMABAD

Assignment #03

GROUP MEMBERS:

BSE173077 - Ajwa Naeem

BSE173029 - Sehar Iftikhar

Section 1

November 2, 2020

Submitted to: Samir Obaid

Table of Contents

1. CASE STUDY

Introduction:
The Automated teller machine (ATM) is an automatic banking machine (ABM) that allows the customer to complete basic transactions without any help from bank representatives. The basic one allows the customer to only draw cash and receive a report of the account balance.

Brief description:
Basically, it is an electronic device that is used by the Bank for transection purpose. The user insert there plastic card which is encoded with the user information on a magnetic strip. The strip contains an identification code that is transmitted to the bank's central computer by modem. The user inserts the plastic card to access the account to access the services provided by the Bank.

There were problems like when the banks were closed at night or you have to write a check before you withdraw your money. Suppose you are facing some problem and you need money urgent and the banks are closed what to do know. To overcome this situation ATM were introduced; they were invented by the Shepherd-Barron in 1960. Now even if banks are closed or its late night everyone can access their money.

The ATM basically, provides many services to the user.

Following are some listed below:

1. Transection of money any time you want.

2. Transfer of money from your account to another.

For the transection of money one only need is to insert card enter pin and then enter the amount he wants. It should be noted that there is a limit of how much money one can withdraw from the account. Because ATM is not for the transection of heavy money but for simple scenarios, and also there must be at least Rs.500 present in the account. If someone thinks that he can withdraw all the money or the amount that is not sufficient in the account he cannot.

Transfer money is another service that ATM provides like you want to send someone money very urgently ATM can help you, no need to worry. To transfer money first you will enter the account number where you want to send money to, the system will verify the account number that weather it exists or not, then you will enter the amount then again the backend program will run to check that weather the entered amount is not less than the current balance and also there must always be Rs.500 present in the account. Then in the last you also have to provide your PIN for verification or say security purpose.

2. FUNCTIONS:

1. PinCheck (string Pin):

This function verifies the pin entered by the account holder. If the pin matches to the pin stored in the database of ATM so system will let him login to the system and allow him to use other services.

2. Transection (double Amount):

This function allows account holder to withdraw money from his account. When account holder requests for the transaction some of the conditions will be checked like; whether the present amount in his account is greater or less.

3. TransferMoney (string Pin, double Amount, string RecAccNum):

This function let he account holder to transfer money to specific account. The parameter RecAccNum (receiver account number) firstly will verify that is this correct account or not. Secondly account holder will enter the amount which he wants to transfer and the system will check if the amount is less than the present amount of the account holder than it will proceed. System will ask the user to enter his pin for verification.

3. BLACK BOX TESTING

3.1. Boundary Value Analysis:

Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values. Total test cases are calculated by the formula 4(n) + 1 (where n is the total number of variables).

Function 1: int PinCheck (string Pin):

Total test cases = 4(1) + 1 = 5.

Input values = min = 0000, min+1 = 0001, normal = 5555, max-1 = 9999, max = 1000.

Case	Pin	Expected output
1	0000	Invalid
2	1111	Valid
3	5555	Invalid
4	9999	Invalid
5	1000	Invalid

Function 2: Void Transaction (double Amount):

Total test cases = 4(1) + 1= 5.

Input values = min = 400, min+1 =500, normal =7000, max-1 =14999, max =15000

Case	Amount	Expected output
1	400	Invalid
2	500	Valid
3	7000	Valid
4	14999	Valid
5	15000	Valid

Function 3: Void TransferMoney (string Pin, double Amount, string RecAccNum):
Total test cases = 4(3) + 1 = 13.

Input values = min =0000, min+1 =1111, normal =5555, max-1 =8888, max =9999

Case	Pin	Amount	RecAccNum	Expected output
1	0000	400	1234	Invalid
2	0000	400	3456	Invalid
3	1111	7000	5678	Valid
4	1111	7000	7898	Valid
5	1111	7000	9898	Valid
6	1111	7000	1234	Valid
7	1111	12000	3456	Valid
8	1111	12000	5678	Valid
9	1111	12000	7898	Valid
10	1111	12000	9898	Valid
11	1111	12000	1234	Valid
12	1111	15000	3456	Valid
13	1111	15000	5678	Valid

3.2. Robust BVA:

Robustness testing is any quality assurance methodology focused on testing the robustness of software. It is a degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions. Total test cases are calculated by the formula 6(n) + 1 (where n is the total number of variables).

Function 1: int PinCheck (string Pin):
Total test cases = 6(1) + 1 = 7.

Input values = min = 0000, min+1 = 0001, normal = 5555, max-1 = 9999, max = 1000.

Case	Pin	Expected output
1	0000	Invalid
2	0001	Valid
3	5555	Invalid
4	9999	Invalid
5	1000	Invalid
6	1001	Invalid

7	1002	Invalid

Function 2: Void Transaction (double Amount):
Total test cases = 6(1) + 1= 7.

Input values = min = 400, min+1 =500, normal =7000, max-1 =14999, max =15000

Case	Amount	Expected output
1	399	Invalid
2	400	Invalid
3	500	Valid
4	7000	Valid
5	14999	Valid
6	15000	Valid
7	150001	Invalid

Function 3: Void TransferMoney (string Pin, double Amount, string RecAccNum):
Total test cases = 6(3) + 1 = 19.

Input values = min =0000, min+1 =1111, normal =5555, max-1 =8888, max =9999

Case	Pin	Amount	RecAccNum	Expected output
1	0000	400	1234	Invalid
2	0000	400	3456	Invalid
3	1111	7000	5678	Valid
4	1111	7000	7898	Valid
5	1111	7000	9898	Valid
6	1111	7000	1234	Valid
7	1111	12000	3456	Valid
8	1111	12000	5678	Valid
9	1111	12000	7898	Valid
10	1111	12000	9898	Valid
11	1111	12000	1234	Valid
12	1111	15000	3456	Valid
13	1111	15000	5678	Valid
14	5555	7000	3456	Valid
15	5555	7000	5678	Valid
16	5555	7000	9898	Valid
17	5555	7000	7898	Valid
18	5555	7000	4567	Valid
19	5555	7000	4978	Valid

3.3. Worst Case BVA testing:

In this testing we test every single possible combination. This type of testing is used in critical systems. Total cases are calculated by the formula 5^n (5 power n) where n is the number of variables. To generate test cases first we choose 5 numbers between the given boundary values (min, min+1, normal, max-1, max).

Function 1: int PinCheck (string Pin):
Total test cases = 5^1 = 5.

Input values = min = 0000, min+1 = 1111, normal = 5555, max-1 = 8888, max = 9999

Case	Pin	Expected output
1	0000	Invalid
2	1111	Valid
3	5555	Invalid
4	8888	Invalid
5	9999	Invalid

Function 2: Void Transaction (double Amount):
Total test cases = 5^1 = 5.

Input values = min = 400, min+1 =500, normal =7000, max-1 =14999, max =15000

Case	Amount	Expected output
1	400	Invalid
2	500	Invalid
3	7000	Valid
4	14999	Valid
5	15000	Valid

Function 3: Void TransferMoney (string Pin, double Amount, string RecAccNum):
Total test cases = 5^3 = 125.

Input values = min =0000, min+1 =1111, normal =5555, max-1 =8888, max =9999

Case	Pin	Amount	RecAccNum	Expected output
1	0000	400	1234	Invalid
2	0000	400	3456	Invalid
3	0000	400	5678	Invalid
4	0000	400	7898	Invalid
5	0000	400	9898	Invalid
6	0000	500	1234	Invalid
7	0000	500	3456	Invalid
8	0000	500	5678	Invalid

9	0000	500	7898	Invalid
10	0000	500	9898	Invalid
11	0000	7000	1234	Invalid
12	0000	7000	3456	Invalid
13	0000	7000	5678	Invalid
14	0000	7000	7898	Invalid
15	0000	7000	9898	Invalid
16	0000	12000	1234	Invalid
17	0000	12000	3456	Invalid
18	0000	12000	5678	Invalid
19	0000	12000	7898	Invalid
20	0000	12000	9898	Invalid
21	0000	14999	1234	Invalid
22	0000	14999	3456	Invalid
23	0000	14999	5678	Invalid
24	0000	14999	7898	Invalid
25	0000	14999	9898	Invalid
26	1111	400	1234	Invalid
27	1111	400	3456	Invalid
28	1111	400	5678	Invalid
29	1111	400	7898	Invalid
30	1111	400	9898	Invalid
31	1111	400	1234	Invalid
32	1111	500	3456	Invalid
33	1111	500	5678	Invalid
34	1111	500	7898	Invalid
35	1111	500	9898	Invalid
36	1111	500	1234	Valid
37	1111	7000	3456	Valid
38	1111	7000	5678	Valid
39	1111	7000	7898	Valid
40	1111	7000	9898	Valid
41	1111	7000	1234	Valid
42	1111	12000	3456	Valid
43	1111	12000	5678	Valid
44	1111	12000	7898	Valid
45	1111	12000	9898	Valid
46	1111	12000	1234	Valid
47	1111	14999	3456	Valid
48	1111	14999	5678	Valid
49	1111	14999	7898	Valid
50	1111	14999	9898	Valid

3.4. Robust Worst Case BVA:

If the function under test were to be of the greatest importance we could use a method named Robust Worst-Case testing. Total test cases are calculated by the formula 7^n (where n is the total number of variables).

Function 1: int Pincheck(string Pin):
Total test cases: 7^1 = 7

Pin = {-1111, 0000, 1111, 5555, 8888, 9898, 9999}.

Case	Pin	Expected Output
1	-1111	Fall
2	0000	Invalid
3	1111	Valid
4	5555	Invalid
5	8888	Invalid
6	9898	Invalid
7	9999	Fall

Function 2: void Transection (string Amount):
Total test cases= 7^1 = 7

Amount = {0, 400, 500, 7000, 12000, 14999, 15000}.

Case	Amount	Expected Output
1	0	Invalid
2	400	Invalid
3	500	Invalid
4	7000	Valid
5	12000	Valid
6	14999	Valid
7	15000	Invalid

Function 3: Void TransferMoney (string Pin, double Amount, string RecAccNum):
Pin = {-1111, 0000, 1111, 5555, 8888, 9898, 9999}.

Amount = {0, 400, 500, 7000, 12000, 14999, 15000}.

RecAccNum = {0000, 1234, 3456, 5678, 7898, 9898, 9999}

Total test cases= 7^3 = 343

Case	**Pin**	**Amount**	**RecAccNum**	**Expected Output**
1	-1111	0000	0000	Invalid
2	-1111	0000	1234	Invalid
3	-1111	0000	3456	Invalid
4	-1111	0000	5678	Invalid
5	-1111	0000	7898	Invalid
6	-1111	0000	9898	Invalid
7	-1111	0000	9999	Invalid
8	-1111	400	0000	Invalid
9	-1111	400	1234	Invalid
10	-1111	400	3456	Invalid
11	-1111	400	5678	Invalid
12	-1111	400	7898	Invalid
13	-1111	400	9898	Invalid
14	-1111	400	9999	Invalid
15	-1111	500	0000	Invalid
16	-1111	500	1234	Invalid
17	-1111	500	3456	Invalid
18	-1111	500	5678	Invalid
19	-1111	500	7898	Invalid
20	-1111	500	9898	Invalid
21	-1111	500	9999	Invalid
22	-1111	7000	0000	Invalid
23	-1111	7000	1234	Invalid
24	-1111	7000	3456	Invalid
25	-1111	7000	5678	Invalid
26	-1111	7000	7898	Invalid
27	-1111	7000	9898	Invalid
28	-1111	7000	9999	Invalid
29	-1111	12000	0000	Invalid
30	-1111	12000	1234	Invalid
31	-1111	12000	3456	Invalid
32	-1111	12000	5678	Invalid
33	-1111	12000	7898	Invalid
34	-1111	12000	9898	Invalid
35	-1111	12000	9999	Invalid
36	-1111	14999	0000	Invalid
37	-1111	14999	1234	Invalid
38	-1111	14999	3456	Invalid
39	-1111	14999	5678	Invalid
40	-1111	14999	7898	Invalid
41	-1111	14999	9898	Invalid
42	-1111	14999	9999	Invalid
43	-1111	15000	0000	Invalid
44	-1111	15000	1234	Invalid

45	-1111	15000	3456	Invalid
46	-1111	15000	5678	Invalid
47	-1111	15000	7898	Invalid
48	-1111	15000	9898	Invalid
49	-1111	15000	9999	Invalid
50	0000	400	0000	Invalid
51	0000	400	1234	Invalid
52	0000	400	3456	Invalid
53	0000	400	5678	Invalid
54	0000	400	7898	Invalid
55	0000	400	9898	Invalid
56	0000	400	9999	Invalid
57	0000	400	0000	Invalid
58	0000	400	1234	Invalid
59	0000	400	3456	Invalid
60	0000	400	5678	Invalid
61	0000	400	7898	Invalid
62	0000	400	9898	Invalid
63	0000	400	9999	Invalid
64	0000	400	0000	Invalid
65	0000	400	1234	Invalid
66	0000	400	3456	Invalid
67	0000	400	5678	Invalid
68	0000	400	7898	Invalid
69	0000	400	9898	Invalid
70	0000	400	9999	Invalid
71	0000	400	0000	Invalid
72	0000	400	1234	Invalid
73	0000	400	3456	Invalid
74	0000	400	5678	Invalid
75	0000	400	7898	Invalid
76	0000	400	9898	Invalid
77	0000	400	9999	Invalid
78	0000	400	0000	Invalid
79	0000	400	1234	Invalid
80	0000	400	3456	Invalid
81	0000	400	5678	Invalid
82	0000	400	7898	Invalid
83	0000	400	9898	Invalid
84	0000	400	9999	Invalid
85	0000	400	0000	Invalid
86	0000	400	1234	Invalid
87	0000	400	3456	Invalid
88	0000	400	5678	Invalid
89	0000	400	7898	Invalid

90	0000	400	9898	Invalid
91	0000	400	9999	Invalid
92	1111	500	0000	Invalid
93	1111	500	1234	Invalid
94	1111	500	3456	Invalid
95	1111	500	5678	Invalid
96	1111	500	7898	Invalid
97	1111	500	9898	Invalid
98	1111	500	9999	Invalid
99	1111	500	0000	Invalid
100	1111	500	1234	Invalid
101	1111	500	3456	Invalid
102	1111	500	5678	Invalid
103	1111	500	7898	Invalid
104	1111	500	9898	Invalid
105	1111	500	9999	Invalid
106	1111	500	0000	Invalid
107	1111	500	1234	Invalid
108	1111	500	3456	Invalid
109	1111	500	5678	Invalid
110	1111	500	7898	Invalid
111	1111	500	9898	Invalid
112	1111	500	9999	Invalid
113	1111	500	0000	Invalid
114	1111	500	1234	Invalid
115	1111	500	3456	Invalid
116	1111	500	5678	Invalid
117	1111	500	7898	Invalid
118	1111	500	9898	Invalid
119	1111	500	9999	Invalid
120	1111	500	0000	Invalid
121	1111	7000	1234	Valid
122	1111	7000	3456	Valid
123	1111	7000	5678	Valid
124	1111	7000	7898	Valid
125	1111	7000	9898	Valid
126	1111	7000	9999	Valid
127	1111	7000	0000	Valid
128	1111	7000	1234	Valid
129	1111	7000	3456	Valid
130	1111	7000	5678	Valid
131	1111	7000	7898	Valid
132	1111	7000	9898	Valid
133	1111	7000	9999	Valid
134	1111	7000	0000	Valid

135	1111	7000	1234	Valid
136	1111	7000	3456	Valid
137	1111	7000	5678	Valid
138	1111	7000	7898	Valid
139	1111	7000	9898	Valid
140	1111	7000	9999	Valid
141	5555	7000	0000	Invalid
142	5555	7000	1234	Invalid
143	5555	7000	3456	Invalid
144	5555	7000	5678	Invalid
145	5555	7000	7898	Invalid
146	5555	7000	9898	Invalid
147	5555	7000	9999	Invalid
18	5555	7000	0000	Invalid
149	5555	7000	1234	Invalid
150	5555	7000	3456	Invalid
151	5555	7000	5678	Invalid
152	5555	7000	7898	Invalid
153	5555	7000	9898	Invalid
154	5555	7000	9999	Invalid
155	5555	7000	0000	Invalid
156	5555	7000	1234	Invalid
157	5555	7000	3456	Invalid
158	5555	7000	5678	Invalid
159	5555	7000	7898	Invalid
160	5555	7000	9898	Invalid
161	5555	7000	9999	Invalid
162	5555	7000	0000	Invalid
163	5555	7000	1234	Invalid
164	5555	7000	3456	Invalid
165	5555	7000	5678	Invalid
166	5555	7000	7898	Invalid
167	5555	7000	9898	Invalid
168	5555	7000	9999	Invalid
169	5555	7000	0000	Invalid
170	5555	7000	1234	Invalid

3.5. Strong Robust Equivalence Class Partitioning:

It is used to form groups of test inputs of similar behavior or nature. Test cases are based on classes, not on every input, thereby reduces the time and efforts required to build large number of test cases. There is no specific formula to calculate number of test cases.

Function 1: int PinCheck (string Pin):
In this case, only one class exists.

Pin>=0000 && pin<=9999

Test cases will be:

Pin = (normal value, upper robust value, minimum robust value)

Pin= 1111, 10000, -1111

Function 2: Void Transaction (double Amount):
In this case, only one class exists.

Amount >= 500 && Amount <= 15000

Test cases will be:

Amount = (normal value, upper robust value, minimum robust value)

Amount = 700, 17000, 400

Function 3: Void TransferMoney (string Pin, double Amount, string RecAccNum):
In this case, there are three variables with 1 input class.

Amount >= 500 && Amount <= 15000

Test cases will be:

Amount = (normal value, upper robust value, minimum robust value)

Amount = 700, 17000, 400

Pin >= 0000 && pin <= 9999

Test cases will be:

Pin = (normal value, upper robust value, minimum robust value)

Pin= 1111, 10000, -1111

RecAccNum >=0000 && RecAccNum <=9999

RecAccNum = (normal value, upper robust value, minimum robust value)

RecAccNum = 8888, 11111, -1111

BY COMBINING ALL:

(Pin normal, Amount normal, RecAccNum Normal)

(1111, 1000, 3333)

(Pin normal, Amount normal, RecAccNum lower robust)

(1111, 1000, -1111)

(Pin normal, Amount lower Robust, RecAccNum Normal)

(1111, 400, 3333)

(Pin lower robust, Amount normal, RecAccNum normal)

(-1111, 1000, 3333)

(Pin lower robust, Amount lower robust, RecAccNum lower robust)

(-1111, 400, -1111)

(Pin normal, Amount normal, RecAccNum upper robust)

(1111, 1000, 10000)

(Pin normal, Amount upper robust, RecAccNum normal)

(1111, 20000, 3333)

(Pin upper robust, Amount normal, RecAccNum normal)

(10000, 1000, 333)

(Pin upper robust, Amount upper robust, RecAccNum upper robust)

(10000, 20000, 10000)

(Pin normal, Amount lower robust, RecAccNum upper robust)

(1111, 400, 100000)

(Pin lower robust, Amount normal, RecAccNum lower robust)

(-1111, 1000, -1111)

(Pin lower robust, Amountupper robust, RecAccNum lower robust)

(-1111, 100000, -1111)

Case	Pin	Amount	RecAccNum	Expected output
1	1111	1000	3333	Valid
2	1111	1000	-1111	Invalid
3	1111	400	3333	Invalid
4	-1111	1000	3333	Invalid
5	-1111	400	-1111	Invalid
6	1111	1000	10000	Invalid
7	1111	20000	3333	Invalid
8	10000	1000	3333	Invalid
9	10000	20000	10000	Invalid
10	111	400	100000	Invalid
11	-1111	1000	-1111	Invalid
12	-1111	10000	-1111	Invalid

3.6. Cause Effect Graphing:

Cause effect graphing (CFG) is black box testing technique, that illustrates behavior of software application and the input factors that are responsible for given behavior.

Transfer money from ATM.

Causes:

C1: Valid Card.

C2: Correct pin.

C3: Amount is valid (amount > 500 and amount < 15000).

C4: Receiver account number is valid.

Effects:

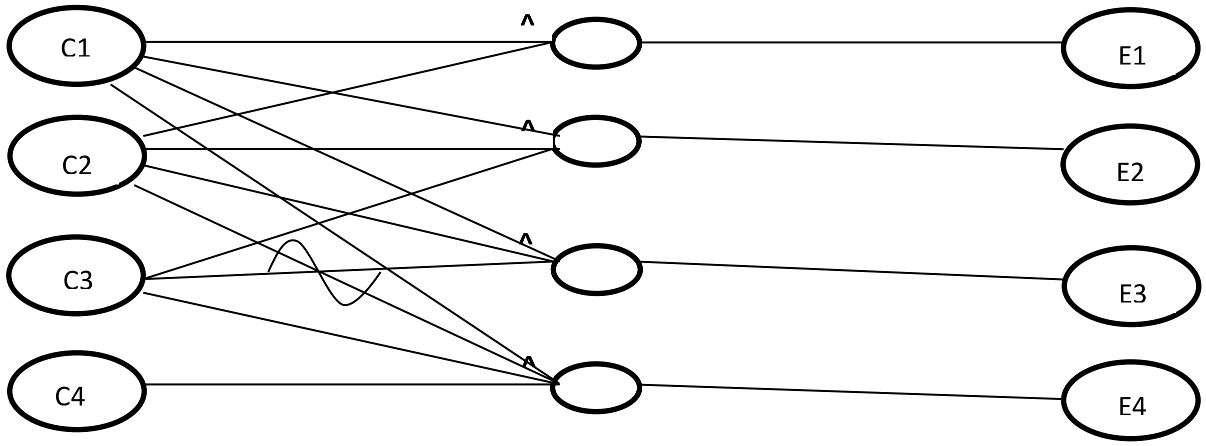
E1: Login

E2: Successfully entered amount.

E3: Current amount is less than amount.

E4: Successfully transferred.

Cause Effect Graph:



Decision Table:

Cause	**T1**	**T2**	**T3**
Valid Card.	1	1	1
Correct pin.	1	1	1
Amount is valid	0	1	1
Receiver account number is valid.	0	0	1
Effects			
Login	1	1	1
Successfully entered amount.	0	1	1
Current amount is less than amount	0	0	1
Successfully transferred.	0	0	1

Test Cases:

Applying robust boundary value analysis.

Test case #	**Inputs (Causes)**			**Expected Output (Effects)**
	Pin	**Amount**	**RecAccNum**	
T1	0000	500	0000	Valid
T2	1111	5001	1111	Valid
T3	4444	2500	1000	Valid
T4	8999	14999	8999	Valid
T5	9999	15000	9999	Valid
T6	-1111	4999	-1111	Invalid
T7	10000	16000	100000	Invalid