CS 4472B Assignment 1 - BlackBox Testing

Department of Computer Science

Western University

February 10st, 2022

Prof. Kostas Kontogiannis

Group 20

Amanpreet Gill

Ositadinma Arimah

Atulpreet Johar

Bohr Deng

Table of Contents

FormatChecker.java

PIN Checker

Requirements/Assumptions

The PIN variable is required to be 4 digits in length. We can create 3 distinct equivalence classes for the length of a PIN, x:

$$\{(x < 3),\ (x > 4),\ (x = 4)\}$$

The values selected for each equivalence class is:

$(x < 3)$— 3 digits long

$(x > 4)$— 5 digits long

$(x = 4)$— 4 digits long

PIN Characters

We can also view each character as an ASCII value, where:

- `"/` " - 47
- `"0"` - 48
- `"9"` - 57
- `":"` - 58
- `" "` - 32

We will refer to the character not the ASCII value from here on, but the thought process is that they can be mapped to sequential int values.

Given each character in the PIN is independent, an invalid digit in any point in the PIN is as good as an invalid digit at any other point in the PIN. This means an invalid input at the first digit spot or the last, are equivalent classes.

In this way, each equivalence class for PIN length can vary one character with the following values:

`min-` - `"/"`

`min` - `"0"`

`min+` - `"1"`

`nominal` - `“3”`

`max-` - `"8"`

`max` - `"9"`

`max+` - `":"`

Test Cases for Weak Robust Equivalence Class Analysis

Test Number	Char 1	Char 2	Char 3	Char 4	Char 5	Expected Value	Pass/Fail
1	3	3	:			Throw Exception	P
2	3	3	9			Throw Exception	P
3	3	3	8			Throw Exception	P
4	3	3	3			Throw Exception	P
5	3	3	1			Throw Exception	P
6	3	3	0			Throw Exception	P
7	3	3	/			Throw Exception	P
8	:	3	3	3		Throw Exception	P
9	9	3	3	3		No Exception	F
10	8	3	3	3		No Exception	P
11	3	3	3	3		No Exception	P
12	1	3	3	3		No Exception	P
13	0	3	3	3		No Exception	P
14	/	3	3	3		Throw Exception	P
15	3	3	3	3	:	Throw Exception	P
16	3	3	3	3	9	Throw Exception	P
17	3	3	3	3	8	Throw Exception	P
18	3	3	3	3	3	Throw Exception	P
19	3	3	3	3	1	Throw Exception	P
20	3	3	3	3	0	Throw Exception	P
21	3	3	3	3	/	Throw Exception	P

Card Checker

Requirements/Assumptions

The credit card checker has 3 dimensions of requirements:

- A card must be valid by Luhn's formula
- Each card has it's own length requirement
- Each card begins with a certain number sequence depending on the issuer

Card Type	Begins With	Length
Visa	4	13 or 16 or 19
Mastercard	51, 52, 53, 54, 55, 222100 or 272099	16
AMEX	34, 37	15
Discover	6011, 622126-622925, 644, 645, 646, 647, 648, 649, 65	16 or 19
Dinners - Carte Blanche	300, 301, 302, 303, 304, 305	14
Dinners - International	36	14
Dinners - Canada & US	54	16
JCB	3528-3589	16 or 19

Assumptions

1. This site (provided in the assignment) is considered to be accurate
2. Luhn's formula will be ignored here for the sake of this assignment
    a. Seems there are different variations to the formula
    b. Using the formula to generate or check card numbers in the test code would introduce a new portion of potentially faulty code
    c. Unsure how to handle this situation

Equivalence Classes

Each card would make up an equivalence class:

$\{AMEX,\ Visa,\ Mastercard,\ Discover,\ Diners\ Club\ -\ CB,\ Diners\ Club\ -\ I,\ Diners\ Club\ -\ UC,\ JCB\}$

The domain of each issuer type's valid set of card numbers is given in the format below:
$Issuer$ - $\{([Starting\ Digits])([Length])\}$

$AMEX$ - $\{(34, 37)(15)\}$
$Diners\ Club(DC)\ -\ CB$ - $\{(300\ -\ 305)(14)\}$
$Diners\ Club(DC)\ -\ I$ - $\{(36)(14)\}$
$Diners\ Club(DC)\ -\ UC$ - $\{(54)(16)\}$
$Discover$ - $\{(6011,\ 622126\ -\ 622925,\ 644\ -\ 649,\ 65)(16,\ 19)\}$
JCB - $\{(3528\ -\ 3589)(16,\ 19)\}$
$Mastercard$ - $\{(51\ -\ 55,\ 222100,\ 272099)(16)\}$
$Visa$ - $\{(4)(13,\ 16,\ 19)\}$

For the issuers with a range of starting values, we can use the ***max***, ***nominal***, and ***min*** of the range. We want to take the cartesian product of the $Starting\ Digits$ and $Length$ values for each equivalence class separately.

Test Cases for Strong Normal Equivalence Analysis

`Length` - Represents the length of the card number. Accounting for the `IIN` a string made up of a repeated digit will round out the card number length to equal `length`.

Test Number	Issuer	IIN (Start With)	Length	Expected Value	Pass/Fail
1	AMEX	34	15	No Exception	P
2	AMEX	37	15	No Exception	P
3	DCCB	300	14	No Exception	P
4	DCCB	303	14	No Exception	P
5	DCCB	305	14	No Exception	P
6	DCI	36	14	No Exception	P
7	DCUC	54	16	No Exception	P
8	Discover	6011	16	No Exception	P
9	Discover	622126	16	No Exception	F
10	Discover	622440	16	No Exception	F
11	Discover	622925	16	No Exception	F

12	Discover	644	16	No Exception	F
13	Discover	647	16	No Exception	F
14	Discover	649	16	No Exception	F
15	Discover	65	16	No Exception	P
16	Discover	6011	19	No Exception	F
17	Discover	622126	19	No Exception	F
18	Discover	622440	19	No Exception	F
19	Discover	622925	19	No Exception	F
20	Discover	644	19	No Exception	F
21	Discover	647	19	No Exception	F
22	Discover	649	19	No Exception	F
23	Discover	65	19	No Exception	F
24	JCB	3528	16	No Exception	P
25	JCB	3555	16	No Exception	P
26	JCB	3589	16	No Exception	P
27	JCB	3528	19	No Exception	F
28	JCB	3555	19	No Exception	F
29	JCB	3589	19	No Exception	F
30	Mastercard	51	16	No Exception	F
31	Mastercard	53	16	No Exception	F
32	Mastercard	55	16	No Exception	F
33	Mastercard	222100	16	No Exception	F
34	Mastercard	272099	16	No Exception	F
35	Visa	4	13	No Exception	P
36	Visa	4	16	No Exception	P
37	Visa	4	19	No Exception	F

CashValidator.java

CashValidatorTest

The Equivalence Classes for each variable include the following:

1. Withdrawal: {(<0], (0-1000], (1000-5000], (5000-10000], (10000-15000], >15000] and are products of 20 and/or 50}
2. Deposit: {(<0], (0-100], (100-500], (500-1000], (1000-5000], >5000]}

The variables selected from each class consists of:

1. Withdrawal
    a. Class (-0): -250
    b. Class (0 -1000): 350
        i. (20 and 50 combo): 670
    c. Class (1000 -5000): 3600
    d. Class (5000 -10000): 5500
        i. (non-product): 5505
    e. Class (10000-15000): 14600
    f. Class (>15000): 45000
2. Deposit
    a. Class (-0): -10
    b. Class (0 -100): 55
        i. (odd): 56
    c. Class (100 -500): 140
    d. Class (500 -1000): 950
        i. (combo): 710
    e. Class (1000-5000): 4960
    f. Class (>5000): 5200

The following test cases derived from Weak Robust Equivalence Class Analysis are as follows:

Withdrawal	Deposit
1. -250	-10
2. 350	55

3. 670	56
4. 3600	140
5. 5500	950
6. 5505	710
7. 14600	4960
8. 45000	5200

Deposit test results:

Expected outcome	**Actual outcome**	**Test result**
Test Case [1]: false	Test Case [1]: true	Fail
Test Case [2]: true	Test Case [2]: false	Fail
Test Case [3]: false	Test Case [3]: false	Pass
Test Case [4]: true	Test Case [4]: true	Pass
Test Case [5]: true	Test Case [5]: true	Pass
Test Case [6]: true	Test Case [6]: true	Pass
Test Case [7]: true	Test Case [7]: true	Pass
Test Case [8]: true	Test Case [8]: true	Pass

Withdrawal test results:

Expected outcome	**Actual outcome**	**Test result**
Test Case [1]: false	Test Case [1]: true	Fail
Test Case [2]: true	Test Case [2]: true	Pass
Test Case [3]: true	Test Case [3]: true	Pass
Test Case [4]: true	Test Case [4]: true	Pass
Test Case [5]: true	Test Case [5]: true	Pass
Test Case [6]: false	Test Case [6]: false	Pass
Test Case [7]: true	Test Case [7]: true	Pass
Test Case [8]: true	Test Case [8]: true	Pass

FeesCalculator.java

FeesCalulatorDepositTest.java

Explanation:

Weak Robust Equivalence Class Analysis was used to implement the eight cases for testing the deposit method of FeesCalculator.java. Weak robust testing tests one variable from each equivalence class, both valid and invalid variables. Consequently, one variable had an invalid value while the others were valid.

Assumptions:

1. accountBalance has a maximum value of 15000.
2. depositAmount has a maximum value of 1000.

The Equivalence Classes for each variable include the following:

1. depositAmount: {(<0], (0-100], (100-500], (500-1000], >1000]}

2. accountBalance: {(<0], (1000-5000], (5000-10000], (10000-15000], >15000]}
3. studentCheck: {(True), (False)}

The variables selected from each class consists of:

1. depositAmount
    a. Class (<0): -10
    b. Class (0 -100): 55
    c. Class (100 -500): 140
    d. Class (500 -1000): 950
    e. Class (>1000): 5200
2. accountBalance
    a. Class (-0): -240
    b. Class (0 -1000): 540
    c. Class (1000 -5000): 3600
    d. Class (5000 -10000): 7500
    e. Class (10000-15000): 13600
    f. Class (>15000): 45000
3. studentCheck
    a. True
    b. False

The eight test cases and their results derived from Weak Robust Equivalence Class Analysis are as follows:

Test Case	depositAmount	accountBalance	studentCheck	Results of Junit Test (Pass/Fail)
1	55	540	True	Pass
2	140	3600	False	Pass
3	950	7500	False	Pass
4	55	13600	False	Pass
5	-10	540	False	Pass
6	5200	540	False	Pass

7	55	-240	False	Pass
8	55	45000	False	Pass

FeesCalulatorWithdrawTest.java

Explanation:

Robust Worst Case Boundary Value Analysis was used to implement the test cases for testing the withdrawal method of FeesCalculator.java. The withdrawAmount variable will derive the boundary value test cases, and since testing is robust, we test both valid and invalid variables.

Assumptions:

1. withdrawalAmount has a minumum value of 0.
2. withdrawalAmount has a maximum value of 10000.

The Equivalence Classes for each variable include the following:

1. withdrawalAmount: {(<0], (0-1000], (1000-10000], >10000]}
2. dayOfWeek: {(Monday-Friday), (Saturday), (Sunday)}
3. studentCheck: {(True), (False)}

The variables selected from each class consists of:

1. withdrawalAmount
    - Class (0-1000]:
        - Min = 0
        - Min- = -1
        - Min+ = 1
        - Nominal = 500
        - Max = 1000
        - Max- = 999
        - Max+ = 10001
    - Class (1000-10000]:
        - Nominal = 5000
        - Max = 10000
        - Max- = 9990
        - Max+ = 10001
2. dayOfWeek

- Class (Monday-Friday): Thursday
- Saturday
- Sunday

3. studentCheck
   - True
   - False

The test cases derived from Robust Worst Case Boundary Analysis includes:

Test Case	**accountBalance**	**dayOfWeek**	**studentCheck**	**Results of Junit Test (Pass/Fail)**
1	-1	Sunday	True	Pass
2	0	Sunday	True	Pass
3	1	Sunday	True	Pass
4	500	Sunday	True	Pass
5	999	Sunday	True	Pass
6	1000	Sunday	True	Pass
7	1001	Sunday	True	Pass
8	5000	Sunday	True	Pass
9	9999	Sunday	True	Pass

10	10000	Sunday	True	Pass
11	10001	Sunday	True	Pass
12	-1	Thursday	True	Pass
13	0	Thursday	True	Pass
14	1	Thursday	True	Pass
15	500	Thursday	True	Pass
16	999	Thursday	True	Pass
17	1000	Thursday	True	Pass
18	1001	Thursday	True	Pass
19	5000	Thursday	True	Pass
20	9999	Thursday	True	Pass
21	10000	Thursday	True	Pass
22	10001	Thursday	True	Pass
23	-1	Saturday	True	Pass
24	0	Saturday	True	Pass
25	1	Saturday	True	Pass
26	500	Saturday	True	Pass

27	999	Saturday	True	Pass
28	1000	Saturday	True	Pass
28	1001	Saturday	True	Pass
30	5000	Saturday	True	Pass
31	9999	Saturday	True	Pass
32	10000	Saturday	True	Pass
33	10001	Saturday	True	Pass
34	-1	Sunday	False	Pass
35	0	Sunday	False	Pass
36	1	Sunday	False	Pass
37	500	Sunday	False	Pass
38	999	Sunday	False	Pass
39	1000	Sunday	False	Pass
40	1001	Sunday	False	Pass
41	5000	Sunday	False	Pass
42	9999	Sunday	False	Pass
43	10000	Sunday	False	Pass

44	10001	Sunday	False	Pass
45	-1	Thursday	False	Pass
46	0	Thursday	False	Pass
47	1	Thursday	False	Pass
48	500	Thursday	False	Pass
49	999	Thursday	False	Pass
50	1000	Thursday	False	Pass
51	1001	Thursday	False	Pass
52	5000	Thursday	False	Pass
53	9999	Thursday	False	Pass
54	10000	Thursday	False	Pass
55	10001	Thursday	False	Pass
56	-1	Saturday	False	Pass
57	0	Saturday	False	Pass
58	1	Saturday	False	Pass
59	500	Saturday	False	Pass
60	999	Saturday	False	Pass

61	1000	Saturday	False	Pass
62	1001	Saturday	False	Pass
63	5000	Saturday	False	Pass
64	9999	Saturday	False	Pass
65	10000	Saturday	False	Pass
66	10001	Saturday	False	Pass

FeesCalulatorTransferTest.java

The Decision table derived from Decision Table Analysis conducted on the transfer method in FeesCalculator.java is as follows:

Condition	**Case 1**	**Case 2**	**Case 3**	**Case 4**	**Case 5**
transferAmount	<100	<100	<100	<100	>=100
fromAccountBalance	<1000	<1000	>=1000	>=1000	<1000
toAccountBalance	<1000	>=1000	>=1000	>=1000	<1000
studentCheck	True	True	True	True	True

Condition	**Case 6**	**Case 7**	**Case 8**	**Case 9**	**Case 10**	**Case 11**

transferAmount	>=100	>=100	>=100	<100	<100	<100
fromAccountBalance	<1000	>=1000	>=1000	<1000	<1000	>=1000
toAccountBalance	>=1000	<1000	>=1000	<1000	>=1000	>=1000
studentCheck	True	True	True	False	False	False

Condition	**Case 12**	**Case 13**	**Case 14**	**Case 15**	**Case 16**
transferAmount	<100	>=100	>=100	>=100	>=100
fromAccountBalance	>=1000	<1000	<1000	>=1000	>=1000
toAccountBalance	>=1000	<1000	>=1000	<1000	>=1000
studentCheck	False	False	False	False	False

The variables selected from each class consists of:

1. transferAmount
    a. Class (<100): 85
    b. Class (>=100): 100
2. fromAccountBalance
    a. Class (>=1000): 1000
    b. Class (<1000): 850
3. toAccountBalance
    a. Class (<1000): 850

    b. Class (>=1000): 1000
4. studentCheck
    a. True
    b. False

There are 16 test cases because number of rows in decision table = number of test cases.

Therefore, the test cases are the following:

Test Case	transferAmount	fromAccountBalance	toAccountBalance	studentCheck	Test Results Pass/Fail
1	85	850	850	True	Pass
2	85	850	1000	True	Pass
3	85	1000	850	True	Pass
4	85	1000	1000	True	Pass
5	100	850	850	True	Pass
6	100	850	1000	True	Pass
7	100	1000	850	True	Pass
8	100	1000	1000	True	Pass
9	85	850	850	False	Pass
10	85	850	1000	False	Pass
11	85	1000	850	False	Pass

12	85	1000	1000	False	Pass
13	100	850	850	False	Pass
14	100	850	1000	False	Pass
15	100	1000	850	False	Pass
16	100	1000	1000	False	Pass