SWEN90006 Software Testing and Reliability

Assignment 1

Student Name:	Lihua Wang
Student ID:	1164051
Coordinator:	Tim Miller
Semester:	2020/S2

School of Computing and Information Systems`

Table of Contents

Task1- Equivalence Partitioning 3

1. Test template tree 3
2. Justification 4

Task2- Equivalence Class Test Cases 4

Task3- Boundary Value Analysis 5

Task4- Boundary Value Analysis Test Suite 11

Task5- Multiple Condition Coverage 13

API Register 13

API Rent 15

API getWeekDays 17

API monthDuration 19

API dayOfWeek 22

Task7 – Comparison 25

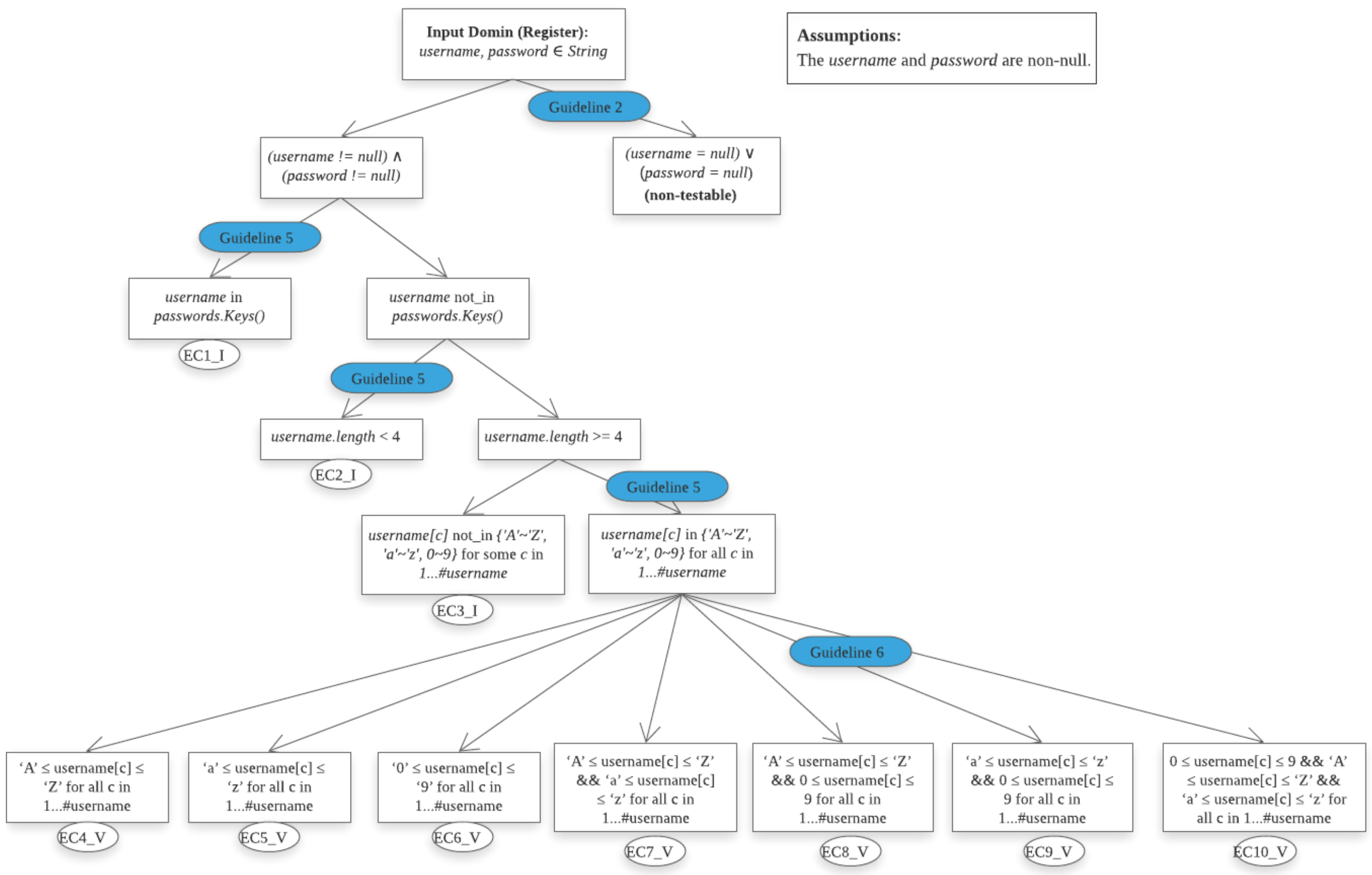
Appendix 25

1. Appendix A-- PartitioningTests.java 25
2. Appendix B-- BoundaryTests.java 37

Task1 - Equivalence Partitioning

1. Test template tree

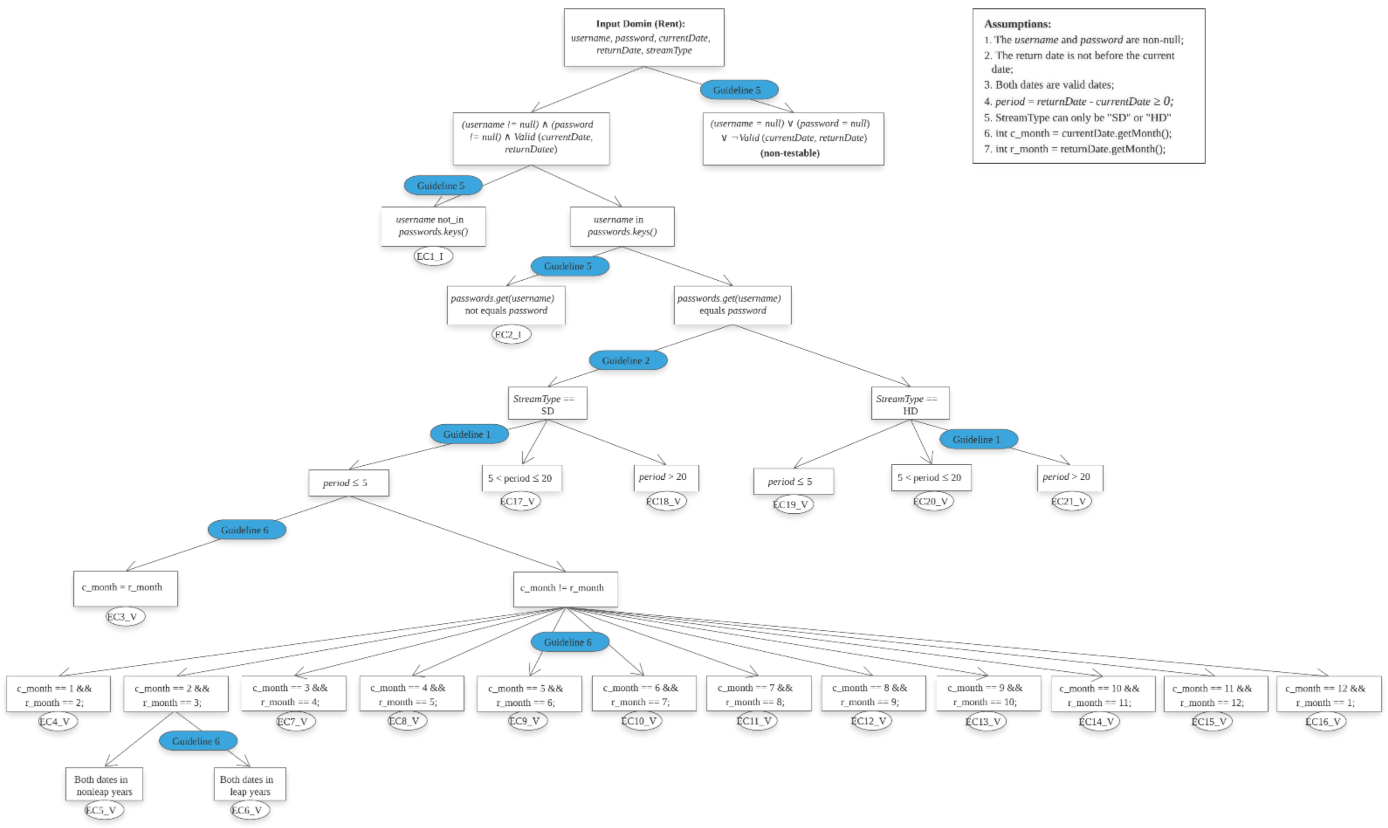
1) Equivalence classes for API: ***register***



Input Domin (Register): *username, password* ∈ *String*
Assumptions: The *username* and *password* are non-null.
Guideline 2
(*username* != *null*) ∧ (*password* != *null*)
(*username* = *null*) ∨ (*password* = *null*) **(non-testable)**
Guideline 5
username in *passwords.Keys()*
EC1_I
username not_in *passwords.Keys()*
Guideline 5
username.length < 4
EC2_I
username.length >= 4
Guideline 5
username[c] not_in {*'A'~'Z', 'a'~'z', 0~9*} for some *c* in *1...#username*
EC3_I
username[c] in {*'A'~'Z', 'a'~'z', 0~9*} for all *c* in *1...#username*
Guideline 6
'A' ≤ username[c] ≤ 'Z' for all c in 1...#username
EC4_V
'a' ≤ username[c] ≤ 'z' for all c in 1...#username
EC5_V
'0' ≤ username[c] ≤ '9' for all c in 1...#username
EC6_V
'A' ≤ username[c] ≤ 'Z' && 'a' ≤ username[c] ≤ 'z' for all c in 1...#username
EC7_V
'A' ≤ username[c] ≤ 'Z' && 0 ≤ username[c] ≤ 9 for all c in 1...#username
EC8_V
'a' ≤ username[c] ≤ 'z' && 0 ≤ username[c] ≤ 9 for all c in 1...#username
EC9_V
0 ≤ username[c] ≤ 9 && 'A' ≤ username[c] ≤ 'Z' && 'a' ≤ username[c] ≤ 'z' for all c in 1...#username
EC10_V

Figure 1 – Test template tree for API-Register

2) Equivalence classes for API: ***Rent***



Input Domin (Rent): username, password, currentDate, returnDate, streamType
Guideline 5
(username != null) ∧ (password != null) ∧ Valid (currentDate, returnDatee)
(username = null) ∨ (password = null) ∨ ¬Valid (currentDate, returnDate) (non-testable)
Guideline 5
username not_in passwords.keys()
EC1_I
username in passwords.keys()
Guideline 5
passwords.get(username) not equals password
EC2_I
passwords.get(username) equals password
Guideline 2
StreamType == SD
StreamType == HD
Guideline 1
period ≤ 5
5 < period ≤ 20
EC17_V
period > 20
EC18_V
Guideline 1
period ≤ 5
EC19_V
5 < period ≤ 20
EC20_V
period > 20
EC21_V
Guideline 6
c_month = r_month
EC3_V
c_month != r_month
Guideline 6
c_month == 1 && r_month == 2;
EC4_V
c_month == 2 && r_month == 3;
c_month == 3 && r_month == 4;
EC7_V
c_month == 4 && r_month == 5;
EC8_V
c_month == 5 && r_month == 6;
EC9_V
c_month == 6 && r_month == 7;
EC10_V
c_month == 7 && r_month == 8;
EC11_V
c_month == 8 && r_month == 9;
EC12_V
c_month == 9 && r_month == 10;
EC13_V
c_month == 10 && r_month == 11;
EC14_V
c_month == 11 && r_month == 12;
EC15_V
c_month == 12 && r_month == 1;
EC16_V
Guideline 6
Both dates in nonleap years
EC5_V
Both dates in leap years
EC6_V

Assumptions:
1. The *username* and *password* are non-null;
2. The return date is not before the current date;
3. Both dates are valid dates;
4. *period = returnDate - currentDate ≥ 0;*
5. StreamType can only be "SD" or "HD"
6. int c_month = currentDate.getMonth();
7. int r_month = returnDate.getMonth();

Figure 2 – Test template tree for API-Rent

2. Justification

Question: Do your set of equivalence classes cover the input space?

Answer: Yes.

Justification: I broke down the input domain by following the guidelines from notes. The equivalence classes for each API in the Xilften program do cover the input space, with 10 equivalence classes for Register, 21 equivalence classes for Rent. All those equivalence classes are disjoint, and the union set of all equivalence classes is each input domain. Besides, in order to create valid test cases efficiently, in rent API, I divided one of the branches in details to detect the possible mutants among the months, they all non-overlapping and coverage for every partition.

Task2 - Equivalence Class Test Cases

The Equivalence Partitioning tests and their corresponding Test IDs are listed below, together with their associated equivalence classes.

Test ID	Test Name	Equivalence Classes

1	duplicateUsername	EC1_I
2	lowerMinLengthUsername	EC2_I
3	illegalCharacterUsername	EC3_I
4	usContains_AZ	EC4_V
5	usContains_az	EC5_V
6	usContains_09	EC6_V
7	usContains_AZ_az	EC7_V
8	usContains_az_09	EC8_V
9	usContains_AZ_09	EC9_V
10	UsContains_AZ_az_09	EC10_V

Table 1 – Equivalence Class Partitioning Test Suite for API Register

Test ID	Test Name	Equivalence Classes
1	usernameNotExists	EC1_I
2	pwdNotMatchUsername	EC2_I
3	SD_PeriodLess5_SameMonth	EC3_V
4	SD_PeriodLess5_C1_R2	EC4_V
5	SD_PeriodLess5_C2_R3_nonleap	EC5_V
6	SD_PeriodLess5_C2_R3_leap	EC6_V
7	SD_PeriodLess5_C3_R4	EC7_V
8	SD_PeriodLess5_C4_R5	EC8_V
9	SD_PeriodLess5_C5_R6	EC9_V
10	SD_PeriodLess5_C6_R7	EC10_V
11	SD_PeriodLess5_C7_R8	EC11_V
12	SD_PeriodLess5_C8_R9	EC12_V
13	SD_PeriodLess5_C9_R10	EC13_V
14	SD_PeriodLess5_C10_R11	EC14_V
15	SD_PeriodLess5_C11_R12	EC15_V
16	SD_PeriodLess5_C12_R1	EC16_V
17	SD_PeriodBetween_5_20	EC17_V
18	SD_PeriodMore_20	EC18_V
19	HD_PeriodLess_5	EC19_V
20	HD_PeriodBetween_5_20	EC20_V
21	HD_PeriodMore_20	EC21_V

Table 2 – Equivalence Class Partitioning Test Suite for API Rent

Task3 - Boundary Value Analysis

For boundary-value analysis, I select test cases of on and off points near the boundary based on the 4 guidelines provided in the notes. The equivalence classes which has no boundary will use the original test case from equivalence partitioning. Otherwise, on point and off point values are picked in each equivalence classes which has a boundary. If more than one equivalence class produce the same on point/off point, there will be only one chosen because that condition could generate the same test case.

The Boundary-value analysis on remaining equivalent classes will be shown in the following table. Note that

1. “EC” refers to “Equivalent Class”,
2. “DUE” refers to “DuplicateUserException”,
3. “IUE” refers to “InvalidUsernameException”,
4. “NSUE” refers to “NoSuchUserException” and
5. “IPE” refers to “IncorrectPasswordException”.

EC	Boundary	Boundary Type	Test Cases Selection	Actual Input	Expected Output
1	username ∩ passwords ≠ null	Inequality, closed	Using Guideline 2: 1. On point: username ∩ passwords = null 2. Off point: username ∩ passwords ≠ null	1. username = “0aAZ”; passwords = <username, p> 2. (inherited) username = “0aAZ”; passwords = <0aAZ, p>	1. Pass 2. DUE
2	username.length < 4	Inequality, open	Using Guideline 2: 1. On point: username.length =4 2. Off point: username.length =3 Using Guideline 4: On point will generate a similar test case as EC1, so we do not select this on point.	2. (inherited) username = “0aA”;	2. IUE
3	username[c] not_in {'A'~'Z', 'a'~'z', 0~9} for some c in 1...#username	Inequality, closed	Using Guideline 3: 1. On point: username contains some illegal characters. 2. Off point: username contains no Illegal characters. Using Guideline 4: The off point is similar to EC1, so we don’t need to consider off point again.	1. (inherited) username = “/0aAZ”	1. IUE
4	‘A’ ≤ username[c] ≤ ‘Z’ for all c in 1...#username	Inequality, closed	Using Guideline 3: For username[c] ≥ ‘A’ 1. On point: username[c] = ‘A’ 2. Off point: username[c] = ‘@’ For username[c] ≤ ‘Z’ 3. On point: username[c] = ‘Z’ 4. Off point: username[c] = ‘[’	(1, 3) username = “ABYZ” (2, 4) username = “@@[[”	(1, 3) Pass (2, 4) IUE
5	‘a’ ≤ username[c] ≤ ‘z’ for all c in 1...#username	Inequality, closed	Using Guideline 3: For username[c] ≥ ‘a’ 1. On point: username[c] = ‘a’ 2. Off point: username[c] = ‘`’ For username[c] ≤ ‘z’ 3. On point: username[c] = ‘z’ 4. Off point: username[c] = ‘{’	(1, 3) username = “abyz” (2, 4) username = “``{{”	(1, 3) Pass (2, 4) IUE
6	‘0’ ≤ username[c]	Inequality,	Using Guideline 3:	(1, 3) username = “0189”	(1, 3) Pass

	≤ ‘9’ for all c in 1...#username	closed	For username[c] ≥ 0 1. On point: username[c] = ‘0’ 2. Off point: username[c] = ‘/’ For username[c] ≤ 9 3. On point: username[c] = ‘9’ 4. Off point: username[c] = ‘:’	(2, 4) username = “//::”	(2, 4) IUE
7	‘A’ ≤ username[c] ≤ ‘Z’ && ‘a’ ≤ username[c] ≤ ‘z’ for some c in 1...#username	Inequality, closed	Using Guideline 3: For username[c] ≥ ‘A’ 1. On point: username[c] = ‘A’ 2. Off point: username[c] = ‘@’ For username[c] ≤ ‘Z’ 3. On point: username[c] = ‘Z’ 4. Off point: username[c] = ‘[’ For username[c] ≥ ‘a’ 5. On point: username[c] = ‘a’ 6. Off point: username[c] = ‘`’ For username[c] ≤ ‘z’ 7. On point: username[c] = ‘z’ 8. Off point: username[c] = ‘{’	(1, 3,5,7) username = “AZaz” (2) username = “@@@@” (4, 6) username = “[[``” (8) username = “{{{{”	(1, 3,5,7) Pass (2) IUE (4, 6) IUE (8) IUE
8	‘A’ ≤ username[c] ≤ ‘Z’ && 0 ≤ username[c] ≤ 9 for some c in 1...#username	Inequality, closed	Using Guideline 3: For username[c] ≥ ‘A’ 1. On point: username[c] = ‘A’ 2. Off point: username[c] = ‘@’ For username[c] ≤ ‘Z’ 3. On point: username[c] = ‘Z’ 4. Off point: username[c] = ‘[’ For username[c] ≥ 0 5. On point: username[c] = ‘0’ 6. Off point: username[c] = ‘/’ For username[c] ≤ 9 7. On point: username[c] = ‘9’ 8. Off point: username[c] = ‘:’	(1, 3,5,7) username = “09AZ” (6) username = “////” (2, 8) username = “: :@@” (4) username = “[[[[”	(1, 3,5,7) Pass (6) IUE (2, 8) IUE (4) IUE
9	‘a’ ≤ username[c] ≤ ‘z’ && 0 ≤ username[c] ≤ 9 for some c in 1...#username	Inequality, closed	Using Guideline 3: For username[c] ≥ ‘a’ 1. On point: username[c] = ‘a’ 2. Off point: username[c] = ‘`’ For username[c] ≤ ‘z’ 3. On point: username[c] = ‘z’ 4. Off point: username[c] = ‘{’ For username[c] ≥ 0 5. On point: username[c] = ‘0’ 6. Off point: username[c] = ‘/’ For username[c] ≤ 9 7. On point: username[c] = ‘9’ 8. Off point: username[c] = ‘:’	(1, 3,5,7) username = “09az” *(6) username = “`////” (2, 8) username = “``::” *(4) username = “{{{{” Using Guideline4: Suites (4) and (6) are similar with EC7-(8) and EC8-(6), so not test again.	(1, 3,5,7) Pass (2, 8) IUE
10	0 ≤ username[c]	Inequality,	Using Guideline 3:	(1, 3, 5, 7, 9, 11)	(1, 3, 5, 7,

	≤ 9 && ‘A’ ≤ username[c] ≤ ‘Z’ && ‘a’ ≤ username[c] ≤ ‘z’ for some c in 1...#username	closed	For username[c] ≥ 0 1. On point: username[c] = ‘0’ 2. Off point: username[c] = ‘/’ For username[c] ≤ 9 3. On point: username[c] = ‘9’ 4. Off point: username[c] = ‘:’ For username[c] ≥ ‘A’ 5. On point: username[c] = ‘A’ 6. Off point: username[c] = ‘@’ For username[c] ≤ ‘Z’ 7. On point: username[c] = ‘Z’ 8. Off point: username[c] = ‘[’ For username[c] ≥ ‘a’ 9. On point: username[c] = ‘a’ 10. Off point: username[c] = ‘`’ For username[c] ≤ ‘z’ 11. On point: username[c] = ‘z’ 12. Off point: username[c] = ‘{’	username = “09AZaz” (2) username = “////” (12) username =”{{{{” (4, 6) username = “: :@@” (8, 10) username = “[[`` Using Guideline4: Suites (2), (12), (4, 6), (8,10) are similar with EC8-(6), EC7-(8), EC8-(2, 8) and EC7-(4, 6) repectively, so not test again.	9, 11) Pass

Table 3 – Boundary Analysis for API Register

EC	Boundary	Boundary Type	Test Cases Selection	Actual Input	Expected Output
1	username ∩ passwords = null	Equality, closed	Using Guideline 1: 1. On point: username ∩ passwords = null 2. Off point: username ∩ passwords ≠ null	1. (inherited) username = “0aAZ”; passwords = <username, p> 2. username = “0aAZ”; passwords = <0aAZ, p>.	1. NSUE 2. Pass
2	passwords.get(username) not equals password	Inequality, closed	Using Guideline 3: 1. On point: password is incorrect 2. Off point: password is correct	1. (inherited) username = “0aAZ”; Password = “pwd”; passwords = <0aAZ, p> 2. username = “0aAZ”; password = “p”; passwords = <0aAZ, p>.	1. IPE 2. Pass
3	{period | StreamType == SD ∧ period ≤ 5 ∧ c_month == r_month}	Equality, closed	Using Guideline 2: 1.on point: period = 5 2.off point: period = 6	1. SD: currentDate= <22,2,2000>; returnDate= <29,2,2000>. 2. SD currentDate= <20,2,2019>; returnDate= <28,2,2019>.	1. cost = 4.0 1. cost = 4.1
4	{period | StreamType == SD ∧ (period ≤ 5) ∧ (c_month == 1 && r_month == 2)}	Inequality, open	Using Guideline 2: 1.on point: period = 5 2.off point: period = 6	1. SD: currentDate= <28,1,2020>; returnDate= <4,2,2020>. 2. SD: currentDate= <28,1, 2019>; returnDate= <5,2, 2019>.	2. cost=4.0 2. cost=4.1
5	{period | StreamType ==	Inequality,	Using Guideline 2:	1. SD:	1. cost=4.0

	SD ∧ (period ≤ 5) ∧ (c_month == 2 && r_month == 3) ∧ nonleap year}	open	1.on point: period = 5 2.off point: period = 6	currentDate= <28,2,2018>; returnDate= <7,3,2018>. 2. SD: currentDate= <28,2, 2019>; returnDate= <10,3, 2019>.	2. cost=4.1
6	{period | StreamType == SD ∧ (period ≤ 5) ∧ (c_month == 2 && r_month == 3) ∧ leap year}	Inequality, open	Using Guideline 2: 1.on point: period = 5 2.off point: period = 6	1. SD: currentDate= <28,2,2020>; returnDate= <6,3,2020>; 2. SD: currentDate= <28,2, 2000>; returnDate= <7, 3, 2000>.	1. cost=4.0 2. cost=4.1
7	{period | StreamType == SD ∧ (period ≤ 5) ∧ (c_month == 3 && r_month == 4)}	Inequality, open	Using Guideline 2: 1.on point: period = 5 2.off point: period = 6	1. SD: currentDate= <28,3,2020>; returnDate= <6,4,2020>. 2. SD: currentDate= <28,3, 2018>; returnDate= <5,4, 2018>.	1. cost=4.0 2. cost=4.1
8	{period | StreamType == SD ∧ (period ≤ 5) ∧ (c_month == 4 && r_month == 5)}	Inequality, open	Using Guideline 2: 1.on point: period = 5 2.off point: period = 6	1. SD: currentDate= <28,4,2000>; returnDate= <5,5,2000>. 2. SD: currentDate= <28,4, 2019>; returnDate= <7,5, 2019>.	1. cost=4.0 2. cost=4.1
9	{period | StreamType == SD ∧ (period ≤ 5) ∧ (c_month == 5 && r_month == 6)}	Inequality, open	Using Guideline 2: 1.on point: period = 5 2.off point: period = 6	1. SD: currentDate= <28,5,2020>; returnDate= <4,6,2020>. 2. SD: currentDate= <28,5, 2018>; returnDate= <5,6, 2018>.	1. cost=4.0 2. cost=4.1
10	{period | StreamType == SD ∧ (period ≤ 5) ∧ (c_month == 6 && r_month == 7)}	Inequality, open	Using Guideline 2: 1.on point: period = 5 2.off point: period = 6	1. SD: currentDate= <28,6,2000>; returnDate= <5,7,2000>. 2. SD: currentDate= <28,6, 2019>; returnDate= <8,7, 2019>.	1. cost=4.0 2. cost=4.1
11	{period | StreamType == SD ∧ (period ≤ 5) ∧ (c_month == 7 && r_month == 8)}	Inequality, open	Using Guideline 2: 1.on point: period = 5 2.off point: period = 6	1. SD: currentDate= <28,7,2020>; returnDate= <4,8,2020>. 2. SD: currentDate= <28,7, 2018>; returnDate= <7,8, 2018>.	1. cost=4.0 2. cost=4.1
12	{period | StreamType == SD ∧ (period ≤ 5) ∧ (c_month == 8 && r_month == 9)}	Inequality, open	Using Guideline 2: 1.on point: period = 5 2.off point: period = 6	1. SD: currentDate= <28,8,2000>; returnDate= <4,9,2000>. 2. SD: currentDate= <28,8, 2019>;	1. cost=4.0 2. cost=4.1

				returnDate= <5,9, 2019>.	
13	{period | StreamType == SD ∧ (period ≤ 5) ∧ (c_month == 9 && r_month == 10)}	Inequality, open	Using Guideline 2: 1.on point: period = 5 2.off point: period = 6	1. SD: currentDate= <28,9,2020>; returnDate= <5,10,2020>. 2. SD: currentDate= <28,9, 2018>; returnDate= <8,10, 2018>.	1. cost=4.0 2. cost=4.1
14	{period | StreamType == SD ∧ (period ≤ 5) ∧ (c_month == 10 && r_month == 11)}	Inequality, open	Using Guideline 2: 1.on point: period = 5 2.off point: period = 6	1. SD: currentDate= <28,10,2000>; returnDate= <6,11,2000>. 2. SD: currentDate= <28,10, 2019>; returnDate= <5,11, 2019>.	1. cost=4.0 2. cost=4.1
15	{period | StreamType == SD ∧ (period ≤ 5) ∧ (c_month == 11 && r_month == 12)}	Inequality, open	Using Guideline 2: 1.on point: period = 5 2.off point: period = 6	1. SD: currentDate= <28,11,2020>; returnDate= <7,12,2020>. 2. SD: currentDate= <28,11, 2018>; returnDate= <6,12, 2018>.	1. cost=4.0 2. cost=4.1
16	{period | StreamType == SD ∧ (period ≤ 5) ∧ (c_month == 12 && r_month == 1)}	Inequality, open	Using Guideline 2: 1.on point: period = 5 2.off point: period = 6	1. SD: currentDate= <28,12,2019>; returnDate= <6,1,2020>. 2. SD: currentDate= <28,12, 2018>; returnDate= <7,1, 2019>.	3. cost=4.0 4. cost=4.1
17	{period | StreamType == SD ∧ (5 < period ≤ 20)}	Inequality, open	Using Guideline 2: For period > 5: 1.on point: period = 5 2.off point: period = 6 For period ≤ 20: 3.on point: period = 20 4. off point: period = 21 Using guideline 4: For the test cases of points on period =5/6 are similar to EC3~15, so we don't need to test again.	3. SD currentDate= <31,1,2020>; returnDate= <29,2,2020>. 4. SD currentDate= <1,3,2018>; returnDate= <31,3,2018>.	3. cost=5.5 4. cost=5.5
18	{period | StreamType == SD ∧ (period > 20)}	Inequality, open	Using Guideline 2: 1.on point: period = 20	N/A	1. N/A

			2.off point: period = 21 Using guideline 4: For the test cases of points on are similar to EC17(3,4), so we don’t need to test again.		
19	{period | StreamType == HD ∧ (period ⩽ 5)}	Inequality, closed	Using Guideline 2: 1.on point: period = 5 2.off point: period = 6	1. HD: currentDate= <28,2,2000>; returnDate= <6,3,2000>. 2. HD: currentDate= <28,8, 2019>; returnDate= <5,9, 2019>.	1. cost=5.0 2. cost=5.1
20	{period | StreamType == HD ∧ (5 < period ≤ 20)}	Inequality, open	Using Guideline 2: For period > 5: 1.on point: period = 5 2.off point: period = 6 For period ≤ 20: 3.on point: period = 20 4. off point: period = 21 Using guideline 4: For the test cases of points on period =5/6 are similar to EC29(3,4), so we don’t need to test again.	3. HD: currentDate= <28,2,2000>; returnDate= <27,3,2000>. 4. HD: currentDate= <2,8, 2018>; returnDate= <31,8, 2018>.	3. cost=6.5 4. cost=6.5
21	{period | StreamType == HD ∧ (period > 20)}	Inequality, open	Using Guideline 2: 1.on point: period = 20 2.off point: period = 21 Using guideline 4: For the test cases of points on are similar to EC17(3,4), so we don’t need to test again.	N/A	N/A

Table 4 – Boundary Analysis for API Rent

Task4 - Boundary Value Analysis Test Suite

Test ID	Test Name	Equivalence Classes
1	usernameNotRegistered	EC1_I, EC2_I
2	(inherit) duplicateUsername	EC1_I
3	(inherit) lowerMinLengthUsername	EC2_I
4	(inherit) illegalCharacterUsername	EC3_I
5	(inherit)usContains_AZ	EC4_V

6	usContains_AZ_off_points	EC4_V
7	(inherit)usContains_az	EC5_V
8	usContains_az_off_points	EC5_V
9	(inherit)usContains_09	EC6_V
10	usContains_09_off_points	EC6_V
11	(inherit)usContains_AZ_az	EC7_V
12	usContains_AZ_az_off_point_1	EC7_V
13	usContains_AZ_az_off_point_2	EC7_V, EC10_V
14	usContains_AZ_az_off_point_3	EC7_V, EC9_V, EC10_v
15	(inherit)usContains_AZ_09	EC8_V
16	usContains_AZ_09_off_point_1	EC8_V, EC9_V, EC10_V
17	usContains_AZ_09_off_point_2	EC8_V, EC10_V
18	usContains_AZ_09_off_point_3	EC8_V
19	(inherit) usContains_az_09	EC9_V
20	usContains_az_09_off_point	EC9_V
21	(inherit) usContains_AZ_az_09	EC10_V

Table 5 – Boundary Test Suite for API Register

Test ID	Test Name	Equivalence Classes
1	usernameNotRegistered	EC1_I, EC2_I
2	usernameNotExists	EC1_I
3	pwdNotMatchUsername	EC2_I
4	(inherit) SD_PeriodLess5_SameMonth	EC3_V
5	SD_PeriodLess5_SameMonth_off_point	EC3_V
6	(inherit) SD_PeriodLess5_C1_R2	EC4_V
7	SD_PeriodLess5_C1_R2_off_point	EC4_V
8	(inherit) SD_PeriodLess5_C2_R3_nonleap	EC5_V
9	SD_PeriodLess5_C2_R3_nonleap_off_point	EC5_V
10	(inherit) SD_PeriodLess5_C2_R3_leap	EC6_V
11	SD_PeriodLess5_C2_R3_leap_off_point	EC6_V
12	(inherit) SD_PeriodLess5_C3_R4	EC7_V
13	SD_PeriodLess5_C3_R4_off_point	EC7_V
14	(inherit) SD_PeriodLess5_C4_R5	EC8_V
15	SD_PeriodLess5_C4_R5_off_point	EC8_V
16	(inherit) SD_PeriodLess5_C5_R6	EC9_V
17	SD_PeriodLess5_C5_R6_off_point	EC9_V
18	(inherit) SD_PeriodLess5_C6_R7	EC10_V
19	SD_PeriodLess5_C6_R7_off_point	EC10_V
20	(inherit) SD_PeriodLess5_C7_R8	EC11_V
21	SD_PeriodLess5_C7_R8_off_point	EC11_V
22	(inherit) SD_PeriodLess5_C8_R9	EC12_V
23	SD_PeriodLess5_C8_R9_off_point	EC12_V
24	(inherit) SD_PeriodLess5_C9_R10	EC13_V
25	SD_PeriodLess5_C9_R10_off_point	EC13_V
26	(inherit) SD_PeriodLess5_C10_R11	EC14_V

27	SD_PeriodLess5_C10_R11_off_point	EC14_V
28	(inherit) SD_PeriodLess5_C11_R12	EC15_V
29	SD_PeriodLess5_C11_R12_off_point	EC15_V
30	(inherit) SD_PeriodLess5_C12_R1	EC16_V
31	SD_PeriodLess5_C12_R1_off_point	EC16_V
32	(inherit) SD_PeriodBetween_5_20	EC17_V, EC18_V
33	SD_PeriodBetween_5_20_off_point	EC17_V, EC18_V
34	(inherit) HD_PeriodLess_5	EC19_V
35	HD_PeriodLess_5_off_point	EC19_V
36	(inherit) HD_PeriodBetween_5_20	EC20_V, EC21_V
37	HD_PeriodBetween_5_20_off_point	EC20_V, EC21_V

Table 6 – boundary Test Suite for API Rent

Task5 - Multiple Condition Coverage

To calculate the multiple-condition coverage, we should clarify all the conditions first and then consider all the combinations of them.

API Register

The following table are the list of all conditions of API-Register, and I labelled in C_n. For C1, C2, C3, there are only TRUE or FALSE conditions. While for C4 have 6 conditions for each one. So the total conditions are 2+2+2+2*2*2*2*=70 objects.

Condition	Branch Code	Permutations	Objective
C1	if (passwords.containsKey(username))	T	1
		F	2
C2	else if (username.length() < MINIMUM_USERNAME_LENGTH)	T	3
		F	4
C3	else if (char c in username.toCharArray())	T	5
		F	6
C4	if (!(('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z') || ('0' <= c && c <= '9')))	{TTTTTT}	7
		……	……
		{FFFFFF}	70

Table 7 –Multiple-condition for API-Register

Equivalence Partitioning

TestID	C1	C2	C3	C4
1	T			
2	F	T		
3	F	F	T	{FTFFFF}

4	F	F	T	{FFTTFF}
5	F	F	T	{TTFFFF}
6	F	F	T	{FFFFTT}
7	F	F	T	{FTTFTF}
8	F	F	T	{FTTTTT}
9	F	F	T	{TTFFTT}
10	F	F	T	{TTTTTT}
Seen	**F, T**	**F, T**	**T**	**…(Total 8)**
Missing	**N/A**	**N/A**	**F**	**…(Total 64-8)**

Table 8 –Partitioning Test Coverage Score for API-Register

After running all test cases from the partitioning test for API Register, C3{F} and C4{…}(56) objectives were unmet. The multiple condition coverage can be calculated with:

$$\frac{\text{objectives met}}{total\ objectives} = \frac{70-56-1}{70} \times 100\% = 18.5\%$$

Boundary Value Analysis

TestID	C1	C2	C3	C4
1	F	F	T	{FTFFFF}
2	T			
3	F	T		
4	F	F	T	{TFFTFT}
5	F	F	T	{FTTTTT}
6	F	F	T	{TTFFTT}
7	F	F	T	{FTTTTT}
8	F	F	T	{FTFFFF}
9	F	F	T	{FTFFFF}
10	F	F	T	{FTTTTT}
11	F	F	T	{FTFFFF}
12	F	F	T	{FTFFFF}
13	F	F	T	{TTFFTT}
14	F	F	T	{FTTTTT}
15	F	F	T	{TTFFTT}
16	F	F	T	{FFTFTT}
17	F	F	T	{FFFFFF}
18	F	F	T	{FTFTFT}
19	F	F	T	{TTFFTT}
20	F	F	T	{TTFFTT}
21	F	F	T	{TTTTTT}
Seen	**T, F**	**T, F**	**T**	**…(total 17)**
Missing	**N/A**	**N/A**	**F**	**…(total 47)**

Table 9 –Boundary Value Test Coverage Score for API-Register

After running all test cases from the boundary test for API Register, 56 out of 70 objectives were unmet. The multiple condition coverage can be calculated with:

$$\frac{\text{objectives met}}{total\ objectives} = \frac{70-47-1}{70} \times 100\% = 31.4\%$$

API Rent

The following table are the list of all conditions of API-Rent, and I labelled in C_n. For C1, C2, C3,C4 they all are only TRUE or FALSE conditions. So the total conditions are 2+2+2+2 = 8 objects.

Condition	Branch Code	Permutations	Objective
C1	if (!passwords.containsKey(username))	T	1
		F	2
C2	else if (!passwords.get(username).equals(password))	T	3
		F	4
C3	if (streamType == StreamType.HD)	T	5
		F	6
C4	if (period > MINIMUM_RENTAL_TIME)	T	7
		F	8

Table 10 –Multiple-condition for API-Rent

Equivalence Partitioning

TestID	C1	C2	C3	C4
1	T			
2	F	T		
3	F	F	F	F
4	F	F	F	F
5	F	F	F	F
6	F	F	F	F
7	F	F	F	F
8	F	F	F	F
9	F	F	F	F
10	F	F	F	F
11	F	F	F	F
12	F	F	F	F
13	F	F	F	F
14	F	F	F	F
15	F	F	F	F
16	F	F	F	F
17	F	F	F	T
18	F	F	F	T

19	F	F	T	F
20	F	F	T	T
21	F	F	T	T
Seen	**T, F**	**T, F**	**T, F**	**T, F**
Missing	**N/A**	**N/A**	**N/A**	**N/A**

Table 11 –Partitioning Test Coverage Score for API-Rent

After running all test cases from the partitioning test for API Rent, 0 out of 8 objectives were unmet. The multiple condition coverage can be calculated with:

$$\frac{\text{objectives met}}{total\ objectives} = \frac{8-0}{8} \times 100\% = 100\%$$

Boundary Value Analysis

TestID	C1	C2	C3	C4
1				
2	T			
3	F	T		
4	F	F	F	F
5	F	F	F	T
6	F	F	F	F
7	F	F	F	T
8	F	F	F	F
9	F	F	F	T
10	F	F	F	F
11	F	F	F	T
12	F	F	F	F
13	F	F	F	T
14	F	F	F	F
15	F	F	F	T
16	F	F	F	F
17	F	F	F	T
18	F	F	F	F
19	F	F	F	T
20	F	F	F	F
21	F	F	F	T
22	F	F	F	F
23	F	F	F	T
24	F	F	F	F
25	F	F	F	T
26	F	F	F	F
27	F	F	F	T
28	F	F	F	F

29	F	F	F	T
30	F	F	F	F
31	F	F	F	T
32	F	F	F	T
33	F	F	F	T
34	F	F	T	F
35	F	F	T	T
36	F	F	T	T
37	F	F	T	T
Seen	**T, F**	**T, F**	**T, F**	**T, F**
Missing	**N/A**	**N/A**	**N/A**	**N/A**

Table 12 –Boundary Value Test Coverage Score for API-Rent

After running all test cases from the boundary test for API Rent, 0 out of 8 objectives were unmet. The multiple condition coverage can be calculated with:

$$\frac{\text{objectives met}}{total\ objectives} = \frac{8-0}{8} \times 100\% = 100\%$$

API getWeekDays

The following table are the list of all conditions of API- getWeekDays, and I labelled in C_n. For C3, C4, C5, there are only TRUE or FALSE conditions. While for C1, C2 have 4 conditions for each one. So the total conditions are 4+4+2+2+2=14 objects.

Condition	Branch Code	Permutations	Objective
C1	If (numberOfWeekDays < MAXIMUM_RENTAL_TIME && !iterantDate.equals(endDate))	{TT}	1
		{TF}	2
		{FT}	3
		{FF}	4
C2	if (dayOfWeek != 0 && dayOfWeek != 6)	{TT}	5
		{TF}	6
		{FT}	7
		{FF}	8
C3	if (day < monthDuration(month, year))	T	9
		F	10
C4	else if (month < 12)	T	11
		F	12
C5	else	T	13
		F	14

Table 13 –Multiple-condition for API-getWeekdays

Equivalence Partitioning

TestID	C1	C2	C3	C4	C5
1					
2					
3	{TT}	{TT}	T		
4	{TT}	{TT}	T		
5	{TT}	{TT}	F	T	
6	{TT}	{TT}	T		
7	{TT}	{TF}	T		
8	{TT}	{TT}	T		
9	{TT}	{TT}	T		
10	{TT}	{FT}	T		
11	{TT}	{TT}	T		
12	{TT}	{TT}	T		
13	{TT}	{TT}	T		
14	{TT}	{TT}	T		
15	{TT}	{TF}	T		
16	{TT}	{TF}	T		
17	{TT}	{TF}	F	T	
18	{TT}	{TT}	F	T	
19	{TT}	{TT}	T		
20	{TT}	{TT}	T		
21	{TT}	{TT}	T		
Seen	**{TT}**	**{TT}{TF}{FT}**	**T, F**	**T**	
Missing	**{TF}{FT}{FF}**	**{FF}**	**N/A**	**F**	**T, F**

Table 14 –Partitioning Test Coverage Score for API-Rent

After running all test cases from the partitioning test for API Re Rent, 7 out 0f 14 objectives were unmet. The multiple condition coverage can be calculated with:

$$\frac{\text{objectives met}}{total\ objectives} = \frac{14-7}{14} \times 100\% = 50\%$$

Boundary Value Analysis

TestID	C1	C2	C3	C4	C5
1					
2					
3					
4	{TT}	{TT}	T		
5	{TT}	{TT}	T		
6	{TT}	{TT}	T		

7	{TT}	{TT}	T		
8	{TT}	{TT}	F	T	
9	{TT}	{TT}	T		
10	{TT}	{TT}	T		
11	{TT}	{TT}	T		
12	{TT}	{TF}	T		
13	{TT}	{TT}	T		
14	{TT}	{TT}	T		
15	{TT}	{FT}	T		
16	{TT}	{TT}	T		
17	{TT}	{TT}	T		
18	{TT}	{TT}	T		
19	{TT}	{TT}	T		
20	{TT}	{TT}	T		
21	{TT}	{TF}	T		
22	{TT}	{TT}	T		
23	{TT}	{TT}	T		
24	{TT}	{TT}	T		
25	{TT}	{TT}	T		
26	{TT}	{TT}	T		
27	{TT}	{TT}	T		
28	{TT}	{TT}	T		
29	{TT}	{TT}	T		
30	{TT}	{TT}	T		
31	{TT}	{TT}	F	F	T
32	{TT}	{TF}	F	T	
33	{TT}	{TT}	T		
34	{TT}	{TT}	T		
35	{TT}	{TT}	T		
36	{TT}	{TT}	T		
37	{TT}	{TT}	T		
Seen	**{TT}**	**{TT}{TF}{FT}**	**T, F**	**T, F**	**T**
Missing	**{TF}{FT}{FF}**	**{FF}**	**N/A**	**N/A**	**F**

Table 15 –Boundary Value Test Coverage Score for API- getWeekdays

After running all test cases from the boundary test for API getWeekdays, 5 out of 14 objectives were unmet. The multiple condition coverage can be calculated with:

$$\frac{\text{objectives met}}{total\ objectives} = \frac{14-5}{14} \times 100\% = 64.28\%$$

API monthDuration

The following table are the list of all conditions of API- monthDuration, and I labelled in C_n. For C2, it only contains TRUE or FALSE conditions. While for C1 have 4 sub-conditions, with each one have a true or false conditions, so total C4 has 2*2*2*2=16 objects. For C3, it also has 4 sub-conditions with true or false conditions, total C3= 2*2*2*2=16, So the total conditions are 16+2+16=34.

Condition	Branch Code	Permutations	Objective
C1	if (month == 2 && (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0)))	T(T(TT))	1
		T(T(TF))	2
		T(T(FT))	3
		T(T(FF))	4
		T(F(TT)	5
		T(F(TF))	6
		T(F(FT))	7
		T(F(FF))	8
		F(T(TT))	9
		F(T(TF))	10
		F(T(FT))	11
		F(T(FF))	12
		F(F(TT)	13
		F(F(TF))	14
		F(F(FT))	15
		F(F(FF))	16
C2	else if (month == 2)	T	17
		F	18
C3	else if (month == 4 || month == 6 || month == 9 || month == 11)	TTTT	19
		TTTF	20
		TTFT	21
		TTFF	22
		TFTT	23
		TFTF	24
		TFFT	25
		TFFF	26
		FTTT	27
		FTTF	28
		FTFT	29
		FTFF	30
		FFTT	31
		FFTF	32
		FFFT	33
		FFFF	34

Table 16 –Multiple-condition for method monthduration

Equivalence Partitioning

TestID	C1	C2	C3
1			

2			
3	T(T(FF))		
4	F(F(TT), T(F(TT)	F	F
5	T(F(FF)), F(F(FF))	T, F	F
6	T(F(TF)), F(F(TF))	T, F	F
7	F(F(TT), F(F(TT)	F, F	F, T
8	F(T(TF)), F(T(TF))	F, F	T, F
9	F(F(TT), F(F(TT)	F, F	F, T
10	F(T(TF)), F(T(TF))	F, F	T, F
11	F(F(TT), F(F(TT)	F, F	F, T
12	F(T(TF)), F(T(TF))	F, F	T, F
13	F(F(TT), F(F(TT)	F, F	F, T
14	F(T(TF)), F(T(TF))	F, F	F, T
15	F(F(TT), F(F(TT)	F, F	T, F
16	F(F(FT)), F(F(TT)	F, F	F, F
17	F(F(TT), T(F(TT)	F	F
18	T(F(FT)), T(F(FT))	T, T	
19	T(F(TF)), F(F(TF))	T, F	F
20	T(T(TF)), F(T(TF))	F	F
21	F(F(FT)), T(F(TT)	F	F
Seen	**T(T(FF)), T(F(FT)), F(F(TT), T(F(TT), F(F(FT)), T(F(TF)), F(F(TF)), F(T(TF)),**	**T, F**	**T, F**
Missing	**TTTT, TTTF, TTFT, TFFF, FTTT, FTFT, FTFF, FFFF**	**N/A**	**N/A**

Table 17 –Partitioning Test Coverage Score for API- monthduration

After running all test cases from the partitioning test for API monthduration, 8 out of 34 objectives were unmet. The multiple condition coverage can be calculated with:

$$\frac{\text{objectives met}}{\textit{total objectives}} = \frac{34-8}{34} \times 100\% = 76.5\%$$

Boundary Value Analysis

TestID	C1	C2	C3
1			
2			
3			
4	T(T(FF))		
5	T(F(FT)), T(F(FT))	T, T	
6	F(F(TT), T(F(TT)	F	F
7	F(F(FT)), T(F(FT))	F, T	F
8	T(F(FT)), F(F(FT))	T, F	F

9	T(F(FT)), F(F(FT))	T, F	F
10	T(F(TF)), F(F(TF))	T, F	F
11	T(F(TT), F(F(TT)	T, F	F
12	F(F(TT), F(F(TT)	F, F	F, T
13	F(F(FT)), F(F(FT))	F, F	F, T
14	F(T(TF)), F(T(TF))	F, F	T, F
15	F(F(FT)), F(F(FT))	F, F	T, F
16	F(F(TT), F(F(TT)	F, F	F, T
17	F(F(FT)), F(F(FT))	F, F	F, T
18	F(T(TF)), F(T(TF))	F, F	T, F
19	F(F(FT)), F(F(FT))	F, F	T, F
20	F(F(TT), F(F(TT)	F, F	F, T
21	F(F(FT)), F(F(FT))	F, F	F, F
22	F(T(TF)), F(T(TF))	F, F	T, F
23	F(F(FT)), F(F(FT))	F, F	F, T
24	F(F(TT), F(F(TT)	F, F	F, T
25	F(F(FT)), F(F(FT))	F, F	T, F
26	F(T(TF)), F(T(TF))	F, F	F, T
27	F(F(FT)), F(F(FT))	F, F	F, T
28	F(F(TT), F(F(TT)	F, F	T, F
29	F(F(FT)), F(F(FT))	F, F	T, F
30	F(F(FT)), F(F(TT)	F, F	F, F
31	F(F(FT)), F(F(FT))	F, F	F, F
32	F(F(TT), T(F(TT)	F	F
33	F(F(FT)), F(F(FT))	F, F	F, F
34	T(F(FT)), T(F(FT))	T, T	
35	T(F(TT), F(F(TT)	F	F
36	T(F(TF)), F(F(TF))	T, F	F
37	F(F(FT)), F(F(FT))	F, F	F, F
Seen	**T(T(FF)), T(F(FT)), F(F(TT), T(F(TT), F(F(FT)), T(F(TF)), F(F(TF)), F(T(TF)),**	**T, F**	**T, F**
Missing	**TTTT, TTTF, TTFT, TFFF, FTTT, FTFT, FTFF, FFFF**	**N/A**	**N/A**

Table 18 –Boundary Value Test Coverage Score for API- monthduration

After running all test cases from the boundary test for API Regi monthduration ster, 8 out of 34 objectives were unmet. The multiple condition coverage can be calculated with:

$$\frac{\text{objectives met}}{\textit{total objectives}} = \frac{34-8}{34} \times 100\% = 76.4\%$$

API dayOfWeek

The following table are the list of all conditions of API- dayOfWeek, and I labelled in C_n. For C1, there are only

TRUE or FALSE conditions. So the total conditions are 2 objects.

Condition	Branch Code	Permutations	Objective
C1	if (month < 3)	T	1
		F	2

Table 19 –Multiple-condition for method dayOFWeek

Equivalence Partitioning

TestID	C1
1	
2	
3	T
4	T
5	T
6	T
7	F
8	F
9	F
10	F
11	F
12	F
13	F
14	F
15	F
16	F
17	T
18	T
19	T
20	T
21	T
Seen	**T, F**
Missing	**N/A**

Table 20 –Partitioning Test Coverage Score for API- dayOFWeek

After running all test cases from the partitioning test for API dayOFWeek, 0 out of 2 objectives were unmet. The multiple condition coverage can be calculated with:

$$\frac{\text{objectives met}}{total\ objectives} = \frac{2-0}{2} \times 100\% = 100\%$$

Boundary Value Analysis

TestID	C1

1	
2	
3	
4	T
5	T
6	T
7	T
8	T
9	T
10	T
11	T
12	F
13	F
14	F
15	F
16	F
17	F
18	F
19	F
20	F
21	F
22	F
23	F
24	F
25	F
26	F
27	F
28	F
29	F
30	F
31	F
32	T
33	F
34	T
35	T
36	T
37	F
Seen	**T, F**
Missing	**N/A**

Table 21 –Boundary Value Test Coverage Score for API- dayOFWeek

After running all test cases from the boundary test for API dayOFWeek, no objectives were unmet. The multiple condition coverage can be calculated with:

$$\frac{\text{objectives met}}{total\ objectives} = \frac{2}{2} \times 100\% = 100\%$$

Task7 – Comparison

For the comparison results of partition test and boundary value division, boundary value division can often achieve better division results. But in fact, the process of boundary value division depends on the equivalence class division, so it is very important to construct an accurate EC. Secondly, when designing test cases, when encountering multivariate divisions, in order to achieve more accurate test results, it may be necessary to subdivide the branches.

In general, the determination of boundary value analysis is more valuable than the equivalent division. The boundary value analysis can be regarded as an equivalent partition expansion. According to the analysis results of the mutants I created, the boundary value can often kill most of the mutants, because the boundary value analysis provides a more complicated set of test cases.

Appendix

1. Appendix A-- PartitioningTests.java

```
package swen90006.xilften;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.*;

public class PartitioningTests
{
	protected Xilften xilften;
	public String username;
	public String password;
	public double expected_cost;
	public double test_cost;

	public PartitioningTests() {
	}

	//Any method annotated with "@Before" will be executed before each test,
	//allowing the tester to set up some shared resources.
	@Before public void setUp()
```

```
{
	xilften = new Xilften();
}

//Any method annotated with "@After" will be executed after each test,
//allowing the tester to release any shared resources used in the setup.
@After public void tearDown() {
}


/**
 * *************** Test method Register ***************
 */

/*EC1_I for method Register*/
@Test(expected = DuplicateUserException.class)
public void duplicateUsername()
		throws Throwable
{
	username = "0aAZ";
	password = "p";

	if(!xilften.isUser(username)) {
		xilften.register(username, password);
	}
	assertTrue("Username is not existed", xilften.isUser(username));
	xilften.register(username, password);
}

/*EC2_I for method Register*/
@Test(expected = InvalidUsernameException.class)
public void lowerMinLengthUsername()
		throws Throwable
{
	username = "0aA";
	password = "p";

	assertFalse("Username is existed", xilften.isUser(username));
	xilften.register(username, password);
}

/*EC3_I for method Register*/
@Test(expected = InvalidUsernameException.class)
public void illegalCharacterUsername()
		throws Throwable
{
	username = "/0aA";
```

```
        password = "p";

        xilften.register(username, password);
        assertFalse("Username is existed", xilften.isUser(username));
}

/*EC4_V for method Register*/
@Test
public void usContains_AZ()
                throws Throwable
{
        username = "ABYZ";
        password = "";

        xilften.register(username, password);
        assertTrue("username can not contain A~Z", xilften.isUser(username));
}

/*EC5_V for method Register*/
@Test
public void usContains_az()
                throws Throwable
{
        username = "abyz";
        password = "p";

        xilften.register(username, password);
        assertTrue("username can not contain a~z", xilften.isUser(username));
}

/*EC6_V for method Register*/
@Test
public void usContains_09()
                throws Throwable
{
        username = "0189";
        password = "pw";

        xilften.register(username, password);
        assertTrue("username can not contain 0~9", xilften.isUser(username));
}

/*EC7_V for method Register*/
@Test
public void usContains_AZ_az()
                throws Throwable
{
        username = "AZaz";
        password = "pw";

        xilften.register(username, password);
        assertTrue("username can not contain a~z && A~Z", xilften.isUser(username));
}

/*EC8_V for method Register*/
@Test
public void usContains_AZ_09()
                throws Throwable
{
        username = "09AZ";
        password = "pw";

        xilften.register(username, password);
        assertTrue("username can not contain A~Z && 0~9", xilften.isUser(username));
}

/*EC9_V for method Register*/
@Test
public void usContains_az_09()
                throws Throwable
{
        username = "09az";
        password = "pw";

        xilften.register(username, password);
        assertTrue("username can not contain a~z && 0~9", xilften.isUser(username));
}

/*EC10_V for method Register*/
@Test
public void usContains_AZ_az_09()
                throws Throwable
{
        username = "09AZaz";
        password = "pw";

        xilften.register(username, password);
        assertTrue("username can not contain A~Z && 0~9 && a~z", xilften.isUser(username));
}

/**
  * *************** Test method Rent ***************
  */
```

```
/*EC1_I for method Rent*/
@Test(expected = NoSuchUserException.class)
public void usernameNotExists()
            throws Throwable
{
      username = "0aAZ";
      password = "p";
      final Date currentDate = new Date(29, 2, 2000);
      final Date returnDate = new Date(29, 2, 2000);

      assertFalse("username: 0aAZ is existed", xilften.isUser(username));
      xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
}

/*EC2_I for method Rent*/
@Test(expected = IncorrectPasswordException.class)
public void pwdNotMatchUsername()
            throws Throwable
{
      username = "0aAZ";
      password = "p";
      final Date currentDate = new Date(28, 2, 2020);
      final Date returnDate = new Date(29, 2, 2020);
      String test_pwd = "pwd";

      if(!xilften.isUser(username)) {
            xilften.register(username, password);
      }
      xilften.rent(username, test_pwd, currentDate, returnDate, Xilften.StreamType.SD);
}

/*EC3_V for method Rent*/
@Test
public void SD_PeriodLess5_SameMonth()
            throws Throwable
{
      username = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
      password = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
      final Date currentDate = new Date(22, 2, 2000);
      final Date returnDate = new Date(29, 2, 2000);
      expected_cost = 4.0;

      if(!xilften.isUser(username)) {
            xilften.register(username, password);
      }
      test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
      assertEquals(expected_cost,test_cost, 0.000000000001);
```

```
}

/*EC4_V for method Rent*/
@Test
public void SD_PeriodLess5_C1_R2()
            throws Throwable
{
        username = "09azAZ";
        password = "";
        final Date currentDate = new Date(28, 1, 2020);
        final Date returnDate = new Date(4, 2, 2020);
        expected_cost = 4.0;

        if(!xilften.isUser(username)) {
                xilften.register(username, password);
        }
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC5_V for method Rent*/
@Test
public void SD_PeriodLess5_C2_R3_nonleap()
            throws Throwable
{
        username = "09azAZ";
        password = " ";
        final Date currentDate = new Date(28, 2, 2018);
        final Date returnDate = new Date(7, 3, 2018);
        expected_cost = 4.0;

        if(!xilften.isUser(username)) {
                xilften.register(username, password);
        }
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC6_V for method Rent*/
@Test
public void SD_PeriodLess5_C2_R3_leap()
            throws Throwable
{
        username = "09azAZ";
        password = "az";
        final Date currentDate = new Date(22, 2, 2100);
        final Date returnDate = new Date(1, 3, 2100);
        expected_cost = 4.0;

        if(!xilften.isUser(username)) {
                xilften.register(username, password);
        }
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC7_V for method Rent*/
@Test
public void SD_PeriodLess5_C3_R4()
                throws Throwable
{
        username = "09azAZ";
        password = "AZ";
        final Date currentDate = new Date(28,3,2020);
        final Date returnDate = new Date(6,4,2020);
        expected_cost = 4.0;

        if(!xilften.isUser(username)) {
                xilften.register(username, password);
        }
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC8_V for method Rent*/
@Test
public void SD_PeriodLess5_C4_R5()
                throws Throwable
{
        username = "09azAZ";
        password = "09";
        final Date currentDate = new Date(28,4,2000);
        final Date returnDate = new Date(5,5,2000);
        expected_cost = 4.0;

        if(!xilften.isUser(username)) {
                xilften.register(username, password);
        }
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC9_V for method Rent*/
@Test
```

```
public void SD_PeriodLess5_C5_R6()
            throws Throwable
{
      username = "09azAZ";
      password = "p";
      final Date currentDate = new Date(28,5,2020);
      final Date returnDate = new Date(4,6,2020);
      expected_cost = 4.0;

      if(!xilften.isUser(username)) {
            xilften.register(username, password);
      }
      test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
      assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC10_V for method Rent*/
@Test
public void SD_PeriodLess5_C6_R7()
            throws Throwable
{
      username = "09azAZ";
      password = "p";
      final Date currentDate = new Date(28,6,2000);
      final Date returnDate = new Date(5,7,2000);
      expected_cost = 4.0;

      if(!xilften.isUser(username)) {
            xilften.register(username, password);
      }
      test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
      assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC11_V for method Rent*/
@Test
public void SD_PeriodLess5_C7_R8()
            throws Throwable
{
      username = "09azAZ";
      password = "p";
      final Date currentDate = new Date(28,7,2020);
      final Date returnDate = new Date(4,8,2020);
      expected_cost = 4.0;

      if(!xilften.isUser(username)) {
            xilften.register(username, password);
```

```
        }
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC12_V for method Rent*/
@Test
public void SD_PeriodLess5_C8_R9()
                throws Throwable
{
        username = "09azAZ";
        password = "p";
        final Date currentDate = new Date(28,8,2000);
        final Date returnDate = new Date(4,9,2000);
        expected_cost = 4.0;

        if(!xilften.isUser(username)) {
                xilften.register(username, password);
        }
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC13_V for method Rent*/
@Test
public void SD_PeriodLess5_C9_R10()
                throws Throwable
{
        username = "09azAZ";
        password = "p";
        final Date currentDate = new Date(28,9,2020);
        final Date returnDate = new Date(5,10,2020);
        expected_cost = 4.0;

        if(!xilften.isUser(username)) {
                xilften.register(username, password);
        }
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC14_V for method Rent*/
@Test
public void SD_PeriodLess5_C10_R11()
                throws Throwable
{
        username = "09azAZ";
        password = "p";
        final Date currentDate = new Date(28,10,2000);
        final Date returnDate = new Date(6,11,2000);
        expected_cost = 4.0;

        if(!xilften.isUser(username)) {
                xilften.register(username, password);
        }
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC15_V for method Rent*/
@Test
public void SD_PeriodLess5_C11_R12()
                throws Throwable
{
        username = "09azAZ";
        password = "p";
        final Date currentDate = new Date(28,11,2020);
        final Date returnDate = new Date(7,12,2020);
        expected_cost = 4.0;

        if(!xilften.isUser(username)) {
                xilften.register(username, password);
        }
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC16_V for method Rent*/
@Test
public void SD_PeriodLess5_C12_R1()
                throws Throwable
{
        username = "09azAZ";
        password = "p";
        final Date currentDate = new Date(28,12,2019);
        final Date returnDate = new Date(6,1,2020);
        expected_cost = 4.0;

        if(!xilften.isUser(username)) {
                xilften.register(username, password);
        }
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
}
```

```
/*EC17_V for method Rent*/
@Test
public void SD_PeriodBetween_5_20()
            throws Throwable
{
      username = "09azAZ";
      password = "p";
      final Date currentDate = new Date(31,1,2020);
      final Date returnDate = new Date(29,2,2020);
      expected_cost = 5.5;

      if(!xilften.isUser(username)) {
            xilften.register(username, password);
      }
      test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
      assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC18_V for method Rent*/
@Test
public void SD_PeriodMore_20()
            throws Throwable
{
      username = "09azAZ";
      password = "p";
      final Date currentDate = new Date(28, 2, 2017);
      final Date returnDate = new Date(29, 2, 2028);
      expected_cost = 5.5;

      if(!xilften.isUser(username)) {
            xilften.register(username, password);
      }
      test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
      assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC19_V for method Rent*/
@Test
public void HD_PeriodLess_5()
            throws Throwable
{
      username = "09azAZ";
      password = "p";
      final Date currentDate = new Date(22, 2, 2100);
      final Date returnDate = new Date(1, 3, 2100);
      expected_cost = 5.0;
```

```
        if(!xilften.isUser(username)) {
            xilften.register(username, password);
        }
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.HD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
    }

    /*EC20_V for method Rent*/
    @Test
    public void HD_PeriodBetween_5_20()
            throws Throwable
    {
        username = "09azAZ";
        password = "p";
        final Date currentDate = new Date(28,2,2000);
        final Date returnDate = new Date(27,3,2000);
        expected_cost = 6.5;

        if(!xilften.isUser(username)) {
            xilften.register(username, password);
        }
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.HD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
    }

    /*EC21_V for method Rent*/
    @Test
    public void HD_PeriodMore_20()
            throws Throwable
    {
        username = "09azAZ";
        password = "p";
        final Date currentDate = new Date(29, 1, 2018);
        final Date returnDate = new Date(29, 2, 2020);
        expected_cost = 6.5;

        if(!xilften.isUser(username)) {
            xilften.register(username, password);
        }
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.HD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
    }

}
```

2. Appendix B-- BoundaryTests.java

```java
package swen90006.xilften;

import java.util.List;
import java.util.ArrayList;
import java.nio.charset.Charset;
import java.nio.file.Path;
import java.nio.file.Files;
import java.nio.file.FileSystems;

import org.junit.*;
import static org.junit.Assert.*;


//By extending PartitioningTests, we inherit tests from the script
public class BoundaryTests
    extends swen90006.xilften.PartitioningTests
{
    /**
     * *************** Test method Register ***************
     */

    /*Test case for method Register:
    EC1_I - on point,
    EC2_I - on point,

    Test case for method Rent:
    EC1_I - off point,
    EC2_I - off point;
    */
    @Test
    public void usernameNotRegistered()
            throws Throwable
    {
        username = "0aAZ";
        password = "p";

        if(!xilften.isUser(username)) {
            xilften.register(username, password);
        }
        assertTrue("Username is not existed", xilften.isUser(username));
    }

    /*EC4_V for method Register*/
    @Test(expected = InvalidUsernameException.class)
```

```
public void usContains_AZ_off_points()
            throws Throwable
{
        username = "@@[[";
        password = "pw";

        xilften.register(username, password);
        assertFalse("username can be @@[[", xilften.isUser(username));
}

/*EC5_V for method Register*/
@Test(expected = InvalidUsernameException.class)
public void usContains_az_off_points()
            throws Throwable
{
        username = "``{{";
        password = "pw";

        xilften.register(username, password);
        assertFalse("username can be ``{{", xilften.isUser(username));
}

/*EC6_V for method Register*/
@Test(expected = InvalidUsernameException.class)
public void usContains_09_off_points()
            throws Throwable
{
        username = "//::";
        password = "pw";

        xilften.register(username, password);
        assertFalse("username can be //::", xilften.isUser(username));
}

/*EC7_V for method Register*/
@Test(expected = InvalidUsernameException.class)
public void usContains_AZ_az_off_point_1()
            throws Throwable
{
        username = "@@@@";
        password = "pw";

        xilften.register(username, password);
        assertFalse("username can be @@@@", xilften.isUser(username));
}

/*EC7_V for method Register*/
```

```
@Test(expected = InvalidUsernameException.class)
public void usContains_AZ_az_off_point_2()
            throws Throwable
{
      username = "[[``";
      password = "pw";

      xilften.register(username, password);
      assertFalse("username can be [[``", xilften.isUser(username));
}

/*EC7_V for method Register*/
@Test(expected = InvalidUsernameException.class)
public void usContains_AZ_az_off_point_3()
            throws Throwable
{
      username = "{{{{";
      password = "pw";

      xilften.register(username, password);
      assertFalse("username can be {{{{", xilften.isUser(username));
}

/*EC8_V for method Register*/
@Test(expected = InvalidUsernameException.class)
public void usContains_AZ_09_off_point_1()
            throws Throwable
{
      username = "////";
      password = "pw";

      xilften.register(username, password);
      assertFalse("username can be ////", xilften.isUser(username));
}

/*EC8_V for method Register*/
@Test(expected = InvalidUsernameException.class)
public void usContains_AZ_09_off_point_2()
            throws Throwable
{
      username = ": :@@";
      password = "pw";

      xilften.register(username, password);
      assertFalse("username can be : :@@", xilften.isUser(username));
}
```

```
/*EC8_V for method Register*/
@Test(expected = InvalidUsernameException.class)
public void usContains_AZ_09_off_point_3()
            throws Throwable
{
        username = "[[[[";
        password = "pw";

        xilften.register(username, password);
        assertFalse("username can be [[[[", xilften.isUser(username));
}

/*EC9_V for method Register*/
@Test(expected = InvalidUsernameException.class)
public void usContains_az_09_off_point()
            throws Throwable
{
        username = "``::";
        password = "pw";

        xilften.register(username, password);
        assertFalse("username can be ``::", xilften.isUser(username));
}


/**
 * *************** Test method Rent ***************
 */

/*EC3_V for method Rent*/
@Test
public void SD_PeriodLess5_SameMonth_off_point()
            throws Throwable
{
        username = "09azAZ";
        password = "";
        final Date currentDate = new Date(20,2,2019);
        final Date returnDate = new Date(28,2,2019);
        expected_cost = 4.1;

        xilften.register(username, password);
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC4_V for method Rent*/
@Test
```

```
public void SD_PeriodLess5_C1_R2_off_point()
            throws Throwable
{
      username = "09azAZ";
      password = "";
      final Date currentDate = new Date(28,1, 2019);
      final Date returnDate = new Date(5,2, 2019);
      expected_cost = 4.1;

      xilften.register(username, password);
      test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
      assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC5_V for method Rent*/
@Test
public void SD_PeriodLess5_C2_R3_nonleap_off_point()
            throws Throwable
{
      username = "09azAZ";
      password = "p";
      final Date currentDate = new Date(28,2, 2019);
      final Date returnDate = new Date(8,3, 2019);
      expected_cost = 4.1;

      xilften.register(username, password);
      test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
      assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC6_V for method Rent*/
@Test
public void SD_PeriodLess5_C2_R3_leap_off_point()
            throws Throwable
{
      username = "09azAZ";
      password = "";
      final Date currentDate = new Date(22,2, 2016);
      final Date returnDate = new Date(1, 3, 2016);
      expected_cost = 4.1;

      xilften.register(username, password);
      test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
      assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC7_V for method Rent*/
```

```
@Test
public void SD_PeriodLess5_C3_R4_off_point()
            throws Throwable
{
      username = "09azAZ";
      password = "";
      final Date currentDate = new Date(28,3, 2018);
      final Date returnDate = new Date(5,4, 2018);
      expected_cost = 4.1;

      xilften.register(username, password);
      test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
      assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC8_V for method Rent*/
@Test
public void SD_PeriodLess5_C4_R5_off_point()
            throws Throwable
{
      username = "09azAZ";
      password = "";
      final Date currentDate = new Date(28,4, 2019);
      final Date returnDate = new Date(7,5, 2019);
      expected_cost = 4.1;

      xilften.register(username, password);
      test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
      assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC9_V for method Rent*/
@Test
public void SD_PeriodLess5_C5_R6_off_point()
            throws Throwable
{
      username = "09azAZ";
      password = "";
      final Date currentDate = new Date(28,5, 2018);
      final Date returnDate = new Date(5,6, 2018);
      expected_cost = 4.1;

      xilften.register(username, password);
      test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
      assertEquals(expected_cost,test_cost, 0.000000000001);
}
```

```java
/*EC10_V for method Rent*/
@Test
public void SD_PeriodLess5_C6_R7_off_point()
            throws Throwable
{
        username = "09azAZ";
        password = "";
        final Date currentDate = new Date(28,6, 2019);
        final Date returnDate = new Date(8,7, 2019);
        expected_cost = 4.1;

        xilften.register(username, password);
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC11_V for method Rent*/
@Test
public void SD_PeriodLess5_C7_R8_off_point()
            throws Throwable
{
        username = "09azAZ";
        password = "";
        final Date currentDate = new Date(28,7, 2018);
        final Date returnDate = new Date(7,8, 2018);
        expected_cost = 4.1;

        xilften.register(username, password);
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC12_V for method Rent*/
@Test
public void SD_PeriodLess5_C8_R9_off_point()
            throws Throwable
{
        username = "09azAZ";
        password = "";
        final Date currentDate = new Date(28,8, 2019);
        final Date returnDate = new Date(5,9, 2019);
        expected_cost = 4.1;

        xilften.register(username, password);
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
}
```

```
/*EC13_V for method Rent*/
@Test
public void SD_PeriodLess5_C9_R10_off_point()
            throws Throwable
{
      username = "09azAZ";
      password = "";
      final Date currentDate = new Date(28,9, 2018);
      final Date returnDate = new Date(8,10, 2018);
      expected_cost = 4.1;

      xilften.register(username, password);
      test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
      assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC14_V for method Rent*/
@Test
public void SD_PeriodLess5_C10_R11_off_point()
            throws Throwable
{
      username = "09azAZ";
      password = "";
      final Date currentDate = new Date(28,10, 2019);
      final Date returnDate = new Date(5,11, 2019);
      expected_cost = 4.1;

      xilften.register(username, password);
      test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
      assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC15_V for method Rent*/
@Test
public void SD_PeriodLess5_C11_R12_off_point()
            throws Throwable
{
      username = "09azAZ";
      password = "";
      final Date currentDate = new Date(28,11, 2018);
      final Date returnDate = new Date(6,12, 2018);
      expected_cost = 4.1;

      xilften.register(username, password);
      test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
      assertEquals(expected_cost,test_cost, 0.000000000001);
```

```
}

/*EC16_V for method Rent*/
@Test
public void SD_PeriodLess5_C12_R1_off_point()
            throws Throwable
{
     username = "09azAZ";
     password = "";
     final Date currentDate = new Date(28,12, 2018);
     final Date returnDate = new Date(7,1, 2019);
     expected_cost = 4.1;

     xilften.register(username, password);
     test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
     assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC17_V, EC18_V for method Rent*/
@Test
public void SD_PeriodBetween_5_20_off_point()
            throws Throwable
{
     username = "09azAZ";
     password = "";
     final Date currentDate = new Date(1,3,2018);
     final Date returnDate = new Date(31,3,2018);
     expected_cost = 5.5;

     xilften.register(username, password);
     test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.SD);
     assertEquals(expected_cost,test_cost, 0.000000000001);
}

/*EC19_V for method Rent*/
@Test
public void HD_PeriodLess_5_off_point()
            throws Throwable
{
     username = "09azAZ";
     password = "";
     final Date currentDate = new Date(22, 2, 2016);
     final Date returnDate = new Date(1, 3, 2016);
     expected_cost = 5.1;

     xilften.register(username, password);
     test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.HD);
```

```
        assertEquals(expected_cost,test_cost, 0.000000000001);
    }

    /*EC20_V, EC21_V for method Rent*/
    @Test
    public void HD_PeriodBetween_5_20_off_point()
            throws Throwable
    {
        username = "09azAZ";
        password = "";
        final Date currentDate = new Date(2,8, 2018);
        final Date returnDate = new Date(31,8, 2018);
        expected_cost = 6.5;

        xilften.register(username, password);
        test_cost = xilften.rent(username, password, currentDate, returnDate, Xilften.StreamType.HD);
        assertEquals(expected_cost,test_cost, 0.000000000001);
    }
}
```