SWEN90006 Software Testing and Reliability

Assignment 1

Semester 2-2019

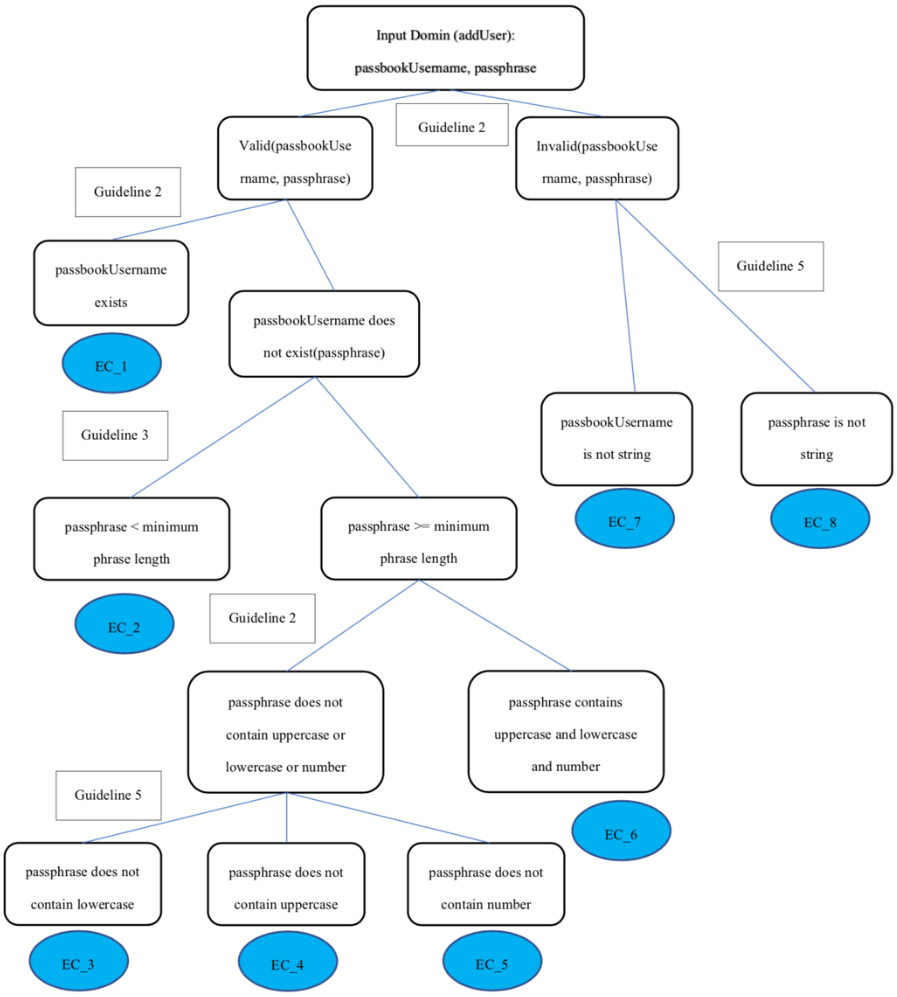
SaiEr Ding 1011802

Task 1

Assumptions:

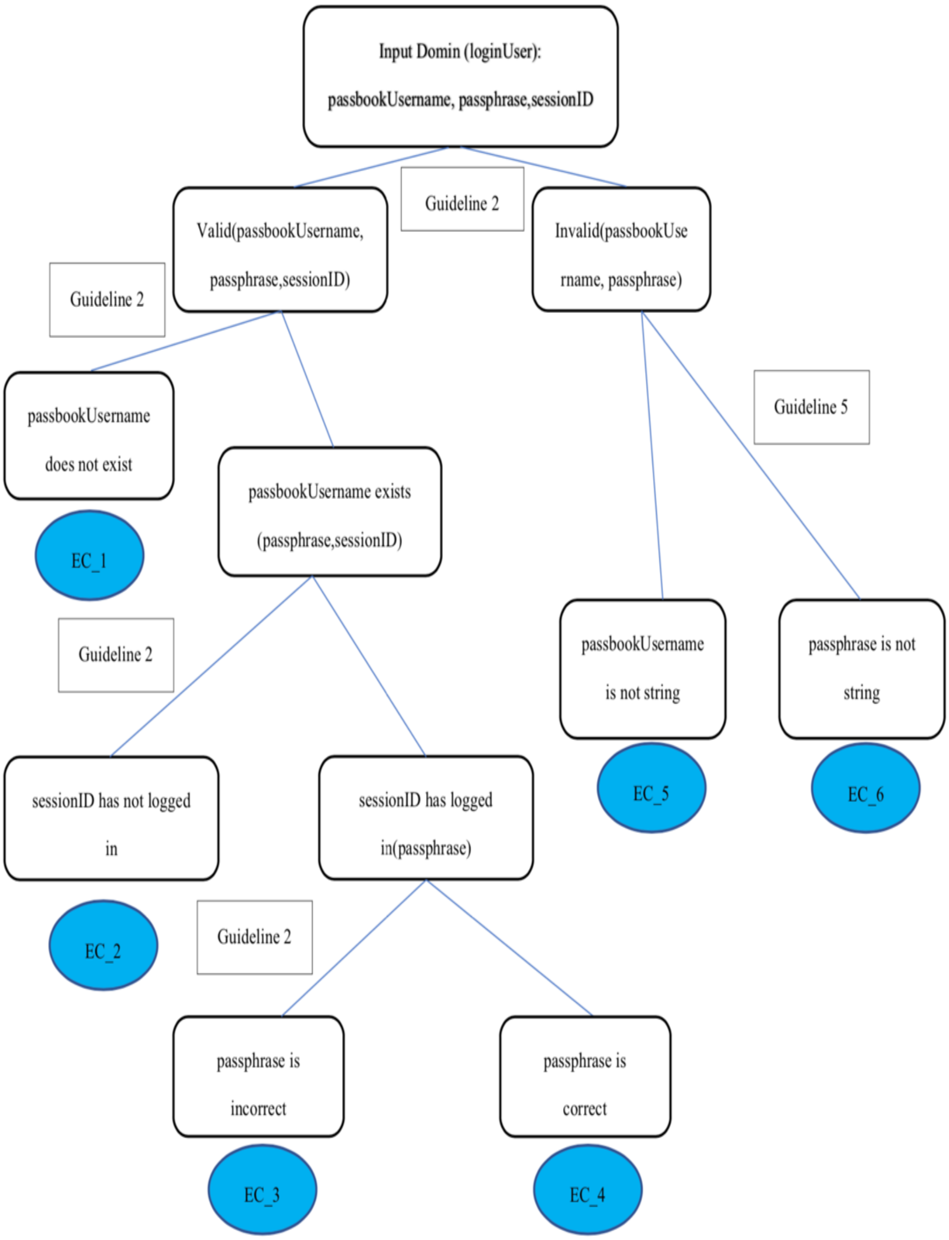
For the first two equivalence classes, the assumptions are “passbookUsername” and “passphrase” are non-null. While for the last two classes, the assumptions are “url” is non-null and “sessionID” is non-null.

Equivalence classes for API: addUser (passbookUsername, passphrase)

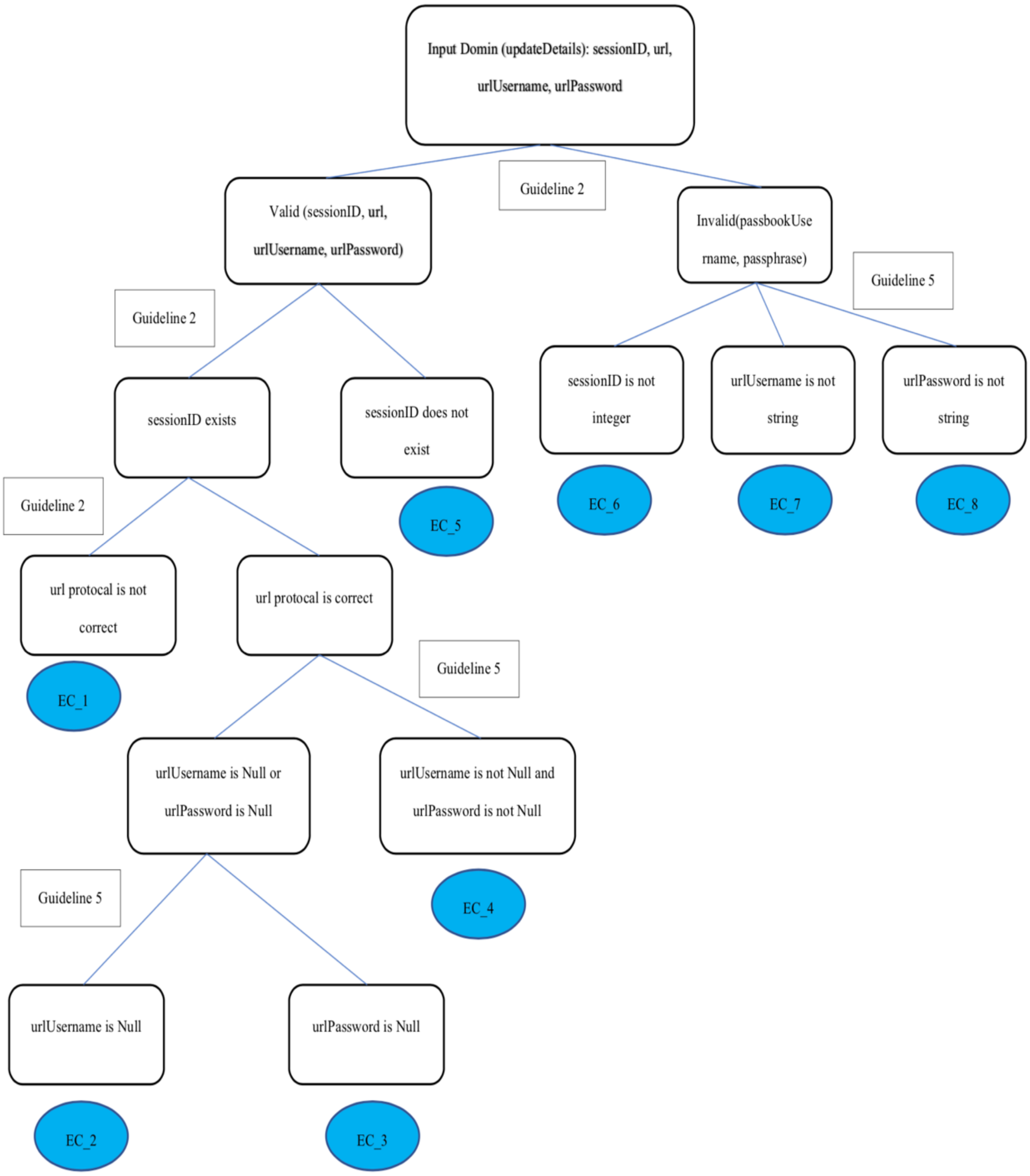


Input Domin (addUser): passbookUsername, passphrase
Guideline 2
Valid(passbookUsername, passphrase)
Invalid(passbookUsername, passphrase)
Guideline 2
passbookUsername exists
EC_1
passbookUsername does not exist(passphrase)
Guideline 5
passbookUsername is not string
EC_7
passphrase is not string
EC_8
Guideline 3
passphrase < minimum phrase length
EC_2
passphrase >= minimum phrase length
Guideline 2
passphrase does not contain uppercase or lowercase or number
passphrase contains uppercase and lowercase and number
EC_6
Guideline 5
passphrase does not contain lowercase
EC_3
passphrase does not contain uppercase
EC_4
passphrase does not contain number
EC_5

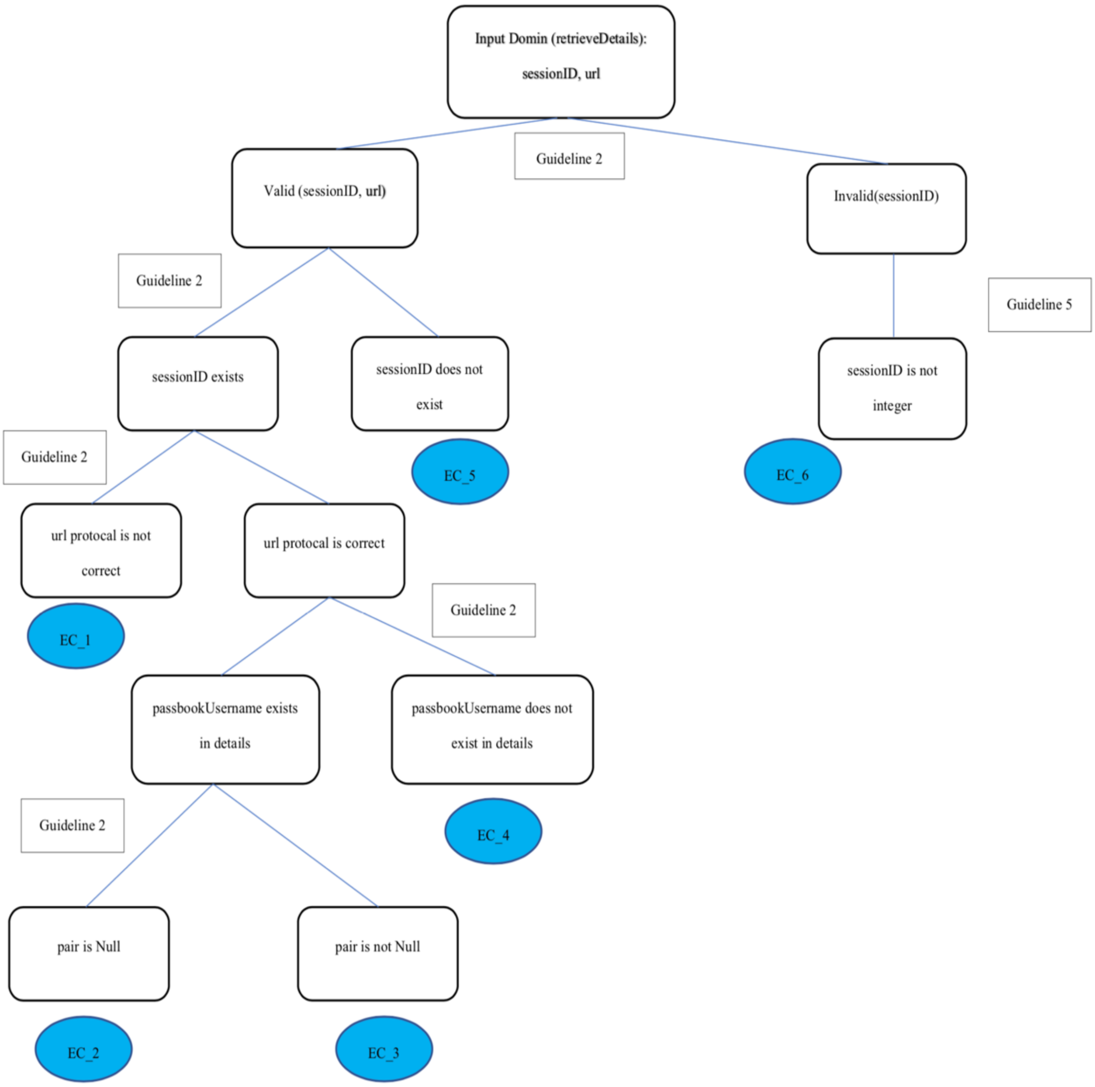
Equivalence classes for API: loginUser (passbookUsername, passphrase, sessionID)


Input Domin (loginUser):
passbookUsername, passphrase,sessionID
Guideline 2
Valid(passbookUsername, passphrase,sessionID)
Invalid(passbookUsername, passphrase)
Guideline 2
passbookUsername does not exist
EC_1
passbookUsername exists (passphrase,sessionID)
Guideline 5
Guideline 2
passbookUsername is not string
passphrase is not string
EC_5
EC_6
sessionID has not logged in
sessionID has logged in(passphrase)
EC_2
Guideline 2
passphrase is incorrect
passphrase is correct
EC_3
EC_4

Equivalence classes for API: updateDetails (urlUsername, urlPassword, sessionID)


Input Domin (updateDetails): sessionID, url, urlUsername, urlPassword
Guideline 2
Valid (sessionID, url, urlUsername, urlPassword)
Invalid(passbookUsername, passphrase)
Guideline 5
Guideline 2
sessionID exists
sessionID does not exist
EC_5
sessionID is not integer
urlUsername is not string
urlPassword is not string
EC_6
EC_7
EC_8
Guideline 2
url protocal is not correct
url protocal is correct
EC_1
Guideline 5
urlUsername is Null or urlPassword is Null
urlUsername is not Null and urlPassword is not Null
EC_4
Guideline 5
urlUsername is Null
urlPassword is Null
EC_2
EC_3

Equivalence classes for API: retrieveDetails (url, sessionID)



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

Answer: Yes.

Justification: The equivalence classes for each API in the passbook program do cover the input space, with 8 equivalence classes for addUser, 6 for loginUser, 8 for updateDetails, and 6 for retrieveDetails. All those equivalence classes are disjoint and the union set of all equivalence classes is each input domain.

Task3

The boundary-value analysis of those four API's equivalence classes are designed below as shown in the tables (Table 3.1 – 3.4).

Table 3.1 Boundary Analysis for API addUser

EC	Boundary	Boundary Type	Test Case Selection	Input	Expected Output
1	passbookUsername ∩ passphrase ≠ Null	Inequality, closed	Using Guideline 2 1. On point: passbookUsername ∩ passphrase = Null 2. Off point: passbookUsername ∩ passphrase ≠ Null	1. string passbookUsername = "dse", string passphrase = "Victording22", int minimum phrase length = 12, map passphrase = <test, test> 2. string passbookUsername = "dse", passphrase = <dse, password>	1. pass 2. throws DuplicateUserException
2	passphrase < minimum phrase length	Inequality, open	Using Guideline 2 1. On point: passphrase = minimum phrase length 2. Off point: passphrase = minimum phrase length - 1 Using Guideline 4: The first test case are similar to EC1, so we don't need to consider that on	2 string passphrase =. "victordin", minimum phrase length = 12	2 throws. WeakPassphraseException

			point and off point again.		
3	passphrase does not contain lowercase	Inequality, closed	Using Guideline 3 1. On point: passphrase has no lowercase 2. Off point: passphrase has a lowercase Using Guideline 4: The second test case are similar to EC1, so we don’t need to consider that on point and off point again.	1. string passphrase = “VICTORDING22”	1. throws WeakPassphraseException
4	passphrase does not contain uppercase	Inequality, closed	Using Guideline 3 1. On point: passphrase has no uppercase 2. Off point: passphrase has a uppercase Using Guideline 4: The second test case are similar to EC1, so we don’t need to consider that on point and off point again.	1. string passphrase = “victording22”	1. throws WeakPassphraseException
5	passphrase does not contain number	Inequality, closed	Using Guideline 3 1. On point: passphrase has no number 2. Off point: passphrase has a number	1. string passphrase = “VICTORDING”,	1. throws WeakPassphraseException

			Using Guideline 4: The second test case are similar to EC1, so we don’t need to consider that on point and off point again.		
6	passphrase contains uppercase and lowercase and number	Inequality, closed	Using Guideline 3 1. On point: passphrase has uppercase, lowercase and number 2. Off point: passphrase has not uppercase or lowercase or number. Using Guideline 4: The first test case are similar to EC1, so we don’t need to consider that on point and off point again.	2. string passphrase = “VICTORD”,	2 throws. WeakPassphraseException
7	passbookUsername is not a string	Inequality, closed	Using Guideline 2 1. On point: passbookUsername is a string 2. Off point: passbookUsername is not a string Using Guideline 4: The first test case are similar to EC1, so we don’t need to consider that on	2 int. passbookUsername= 11	2 throws. WrongTypeException

			point and off point again.		
8	passphrase is not a string	Inequality, closed	Using Guideline 2 1. On point: passphrase is a string 2. Off point: passphrase is not a string Using Guideline 4: The first test case are similar to EC1, so we don’t need to consider that on point and off point again.	2 int passphrase = 11	2 throws. WrongTypeException

Table 3.2 Boundary Analysis for API loginUser

EC	Boundary	Boundary Type	Test Case Selection	Input	Expected Output
1	passbookUsername ∩ passphrase = Null	Equality, closed	Using Guideline 1 1. On point: passbookUsername ∩ passphrase = Null 2. Off point: passbookUsername ∩ passphrase ≠ Null	1. string passbookUsername = “dse”, string passphrase = “Victording22”, map passphrase = <dse23, Victording22> 2. int sessionID = 2, string passbookUsername = “dse”, string passphrase = “Victording22”, map	1. throws NoSuchUserException 2. pass

				passphrase = <dse, Victording22> map sessionID = <dse, 1>	
2	passbookUsername ∩ sessionID ≠ Null	Inequality, open	Using Guideline 2 1. On point: passbookUsername ∩ sessionID = Null 2. Off point: passbookUsername ∩ sessionID ≠ Null Using Guideline 4: The first test case is similar to EC1, so we don’t need to consider that on point and off point again.	2 int sessionID = 2, string passbookUsername = “dse”, string passphrase = “Victording22”, map passphrase = <dse, Victording22> map sessionID = <dse, 2>	2 throws. AlreadyLoggedInException
3	passphrase is incorrect	Inequality, closed	Using Guideline 3 1. On point: passphrase is incorrect 2. Off point: passphrase is correct Using Guideline 4: The second test case is similar to EC1, so we don’t need to consider that on point and off point again.	1. string passbookUsername = “dse”, passphrase = “Victording22” passphrase map = <dse, dse5>	1. throws IncorrectPassphraseException
4	passphrase is correct	Inequality, closed	Using Guideline 3 1. On point: passphrase is correct	N/A	N/A

			2. Off point: passphrase is incorrect Using Guideline 4: Both the first and second test case are similar to EC3, so we don’t need to consider that on point and off point again.		
5	passbookUsername is not a string	Inequality, closed	Using Guideline 2 1. On point: passbookUsername is a string 2. Off point: passbookUsername is not a string Using Guideline 4: The first test case is similar to EC1, so we don’t need to consider that on point and off point again.	2 int passbookUsername = 11	2 throws. WrongTypeException
6	passphrase is not a string	Inequality, closed	Using Guideline 2 1. On point: passphrase is a string 2. Off point: passphrase is not a string Using Guideline 4: The first test case is similar to EC1, so we don’t need to consider that on	2 int passphrase= 11	2 throws WrongTypeException

			point and off point again.		

Table 3.3 Boundary Analysis for API updateDetails

EC	Boundary	Boundary Type	Test Case Selection	Input	Expected Output
1	url protocal ∉ VALID_URL_PROTOCOLS	Inequality, closed	Using Guideline 3 1. On point: url protocal ∉ VALID_URL_PROTOCOLS 2. Off point: url protocal ∈ VALID_URL_PROTOCOLS Using Guideline 4: The second test case is similar to EC5, so we don't need to consider that on point and off point again.	1. string url = "hppp://123.com"	1. throws MalformedURLException
2	urlUsername = Null	Equality, closed	Using Guideline 1 1. On point: urlUsername = Null 2. Off point: urlUsername ≠ Null Using Guideline 4: The second test case is similar to EC5, so we don't need to consider that on point and off point again.	1. string urlUsername = Null, string urlPassword = "Victording22"	1. remove url

3	urlPassword = Null	Equality, closed	Using Guideline 1 1. On point: urlPassword = Null 2. Off point: urlPassword ≠ Null Using Guideline 4: The second test case is similar to EC5, so we don’t need to consider that on point and off point again.	1. string urlUsername = “dse”, string urlPassword = Null	1. remove url
4	urlUsername ≠ Null and urlPassword ≠ Null	Inequality, closed	Using Guideline 2 1. On point: urlUsername ≠ Null and urlPassword ≠ Null 2. Off point: urlUsername = Null or urlPassword = Null Using Guideline 4 Both the first and second test cases are similar to the EC3 and EC2	N/A	N/A
5	sessionID does not exist	Inequality, closed	Using Guideline 3 1. On point: sessionID does not exist 2. Off point: sessionID exists	1. int sessionID = 3, url = http://123.com string urlUsername = “dse”, string urlPassword = “Victording22”, map = <1, dse> 2. int sessionID = 3, url = http://123.com string urlUsername =	1. throws InvalidSessionIDException 2. pass

				"dse", string urlPassword = "Victording22", map = <3, dse>	
6	sessionID is not a integer	Inequality, closed	Using Guideline 2 1. On point: sessionID is a integer 2. Off point: sessionID is not a integer Using Guideline 4: The first test case is similar to EC5, so we don't need to consider that on point and off point again.	2 string sessionID = "VICTORding22"	2 throws WrongTypeException
7	urlUsername is not a string	Inequality, closed	Using Guideline 2 1. On point: urlUsername is a string 2. Off point: urlUsername is not a string Using Guideline 4: The first test case is similar to EC5, so we don't need to consider that on point and off point again.	2 int urlUsername = 11	2 throws WrongTypeException
8	urlPassword is not a string	Inequality, closed	Using Guideline 2 1. On point: urlPassword is a string	2 int urlPassword = 11	2 throws WrongTypeException

			2. Off point: urlPassword is not a string Using Guideline 4: The first test case is similar to EC5, so we don't need to consider that on point and off point again.		

Table 3.4 Boundary Analysis for API retrieveDetails

EC	Boundary	Boundary Type	Test Case Selection	Input	Expected Output
1	url protocal ∉ VALID_URL_PROTOCOLS	Inequality, closed	Using Guideline 3 1. On point: url protocal ∉ VALID_URL_PROTOCOLS 2. Off point: url protocal ∈ VALID_URL_PROTOCOLS Using Guideline 4: The second test case is similar to EC5, so we don't need to consider that on point and off point again.	1. string url = "hppp://123.com"	1. throws MalformedURLException
2	pair = Null	Equality, closed	Using Guideline 1 1. On point: pair = Null 2. Off point: pair ≠ Null Using Guideline 4:	1. pair = <Null,Null>, passbookUsername = "dse" passwordTable = <url, pair>, map = <dse, passwordTable>	1. throws NoSuchURLException

			The second test case is similar to EC5, so we don’t need to consider that on point and off point again.		
3	pair ≠ Null	Inequality, closed	Using Guideline 1 1. On point: pair ≠ Null 2. Off point: pair = Null Using Guideline 4 Both the first and second test cases are similar to the EC2	N/A	N/A
4	passbookUsername ∉ details	Inequality, closed	Using Guideline 3 1. On point: passbookUsername ∉ details 2. Off point: passbookUsername ∈ details Using Guideline 4: The second test case is similar to EC5, so we don’t need to consider that on point and off point again.	1. passbookUsername = “dse” passwordTable = <url, pair>, details = <test, passwordTable>	1. throws NoSuchURLException
5	sessionID does not exist	Inequality, closed	Using Guideline 3 1. On point: sessionID does not exist	1. int sessionID = 3, url = http://123.com , passbookUsername = “dse”, map = <1,dse>, pair =	1. throws InvalidSessionIDException 2. pass

			2. Off point: sessionID exists	<test,pass>, passwordTable = <url, pair>, details = <dse, passwordTable> 2. int sessionID = 3, url = http://123.com , passbookUsername = “dse”, map = <3,dse>, pair = <test,pass>, passwordTable = <url, pair>, details = <dse, passwordTable>	
6	sessionID is not a integer	Inequality, closed	Using Guideline 2 1. On point: sessionID is a integer 2. Off point: sessionID is not a integer Using Guideline 4: The first test case is similar to EC5, so we don’t need to consider that on point and off point again.	2 string sessionID = “VICTORding22”	2 throws WrongTypeException

Task 5

1. To calculate the multiple-condition coverage, we should clarify all the conditions first and then consider all the combinations of them. For the API addUser, all the conditions are listed below (labelled from A to G).

```
A: if (passphrases.containsKey(passbookUsername))
    B: if (passphrase.length() < MINIMUM_PASSPHRASE_LENGTH)
    C: if (i < passphrase.length())
        D: if ('a' <= passphrase.charAt(i) && passphrase.charAt(i) <= 'z')
        E: else if ('A' <= passphrase.charAt(i) && passphrase.charAt(i) <= 'Z')
        F: else if ('0' <= passphrase.charAt(i) && passphrase.charAt(i) <= '9')
    G: if (!containsLowerCase || !containsUpperCase || !containsNumber)
```

And for A,B,C, there are only TRUE or FALSE conditions. While for D,E,F there are four conditions for each one. They are
<D1,(TRUE,TRUE)><D2,(TRUE,FALSE)><D3,(FALSE,TRUE)><D4,(FALSE,FALSE)>
<E1,(TRUE,TRUE)><E2,(TRUE,FALSE)><E3,(FALSE,TRUE)><E4,(FALSE,FALSE)>
<F1,(TRUE,TRUE)><F2,(TRUE,FALSE)><F3,(FALSE,TRUE)><F4,(FALSE,FALSE)> And for G, there are eight conditions.
<G1,(TRUE,TRUE,TRUE)><G2,(FALSE,TRUE,TRUE)><G3,(TRUE,FALSE,TRUE)><G4,(TRUE,TRUE,FALSE)><G5,(FALSE,FALSE,TRUE)><G6,(FALSE,TRUE,FALSE)><G7,(TRUE,FALSE,FALSE)><G8,(FALSE,FALSE,FALSE)>
So that, there are 2 × 3 + 3 × 4 + 8 = 26 conditions in total.

The multiple-condition coverage score of API addUser for the partitioning test is shown in Table 5.1a and the boundary value test is shown in Table 5.1b.

Table 5.1a. Partitioning Test Coverage Score for addUser

EC	TRUE	FALSE
1	A	
2	B	A
3	C E1 F1 G7	A B D1
4	C D1 F1 G6	A B E1
5	C E1 G3	A B D1 F1
6	C D1 E1 F1	A B G8
7	N/A	N/A
8	N/A	N/A

Table 5.1b. Boundary Value Test Coverage Score for addUser

EC	TRUE	FALSE
1.1	C D1 G2	A B E1 F1
1.2	A	
2.2	B	A

3.1	C E1 F1 G7	A B D1
4.1	C D1 F1 G6	A B E1
5.1	C E1 G3	A B D1 F1
6.2	C E1 G3	A B D1 F1
7.2	N/A	N/A
8.2	N/A	N/A

After running all test cases from the partitioning test for API addUser, the true test objectives met includes {A, B, C, D1, E1, F1, G3, G6, G7}, and the false objectives met includes {A, B, D1, E1, F1, G8}, so that the multiple-condition coverage = $\frac{\text{objectives met}}{total\ objectives}$ = $\frac{9+6}{26} \times 100\% = 58\%$

While for the boundary value test for API addUser, the true test objectives met includes {A, B, C, D1, E1, F1, G2, G3, G6, G7}, and the false objectives met includes {A, B, D1, E1, F1}, so that the multiple-condition coverage = $\frac{\text{objectives met}}{total\ objectives}$ = $\frac{10+5}{26}$ $\times 100\% = 58\%$

2. For the API loginUser, all the conditions are listed below (labelled from A to D).

A: if (!passphrases.containsKey(passbookUsername))

B: else if (sessionIDs.get(passbookUsername) != null)

C: else if (!passphrases.get(passbookUsername).equals(passphrase))

D: if (userIDs.containsKey(sessionID))

Every situation has two different conditions (true or false), so that there are 2 × 4 = 8 conditions in total. Then, the multiple-condition coverage score of API loginUser for the partitioning test is shown in Table 5.2a and that for the boundary value test is shown in Table 5.2b.

Table 5.2a. Partitioning Test Coverage Score for loginUser

EC	TRUE	FALSE
1	A	
2	B	A C
3	C	A B
4		A B C D
5	N/A	N/A
6	N/A	N/A

Table 5.2b. Boundary Value Test Coverage Score for loginUser

EC	TRUE	FALSE
1.1	A	
1.2		A B C D
2.2	B	A
3.1	C	A B
5.2	N/A	N/A

6.2	N/A	N/A

After running all test cases from the partitioning test for API loginUser, the true test objectives met includes {A, B, C}, and the false objectives met includes {A, B, C, D}, so that the multiple-condition coverage = $\frac{\text{objectives met}}{total\ objectives}$ = $\frac{3+4}{8}$ $\times 100\% = 87.5\%$. While for the boundary value test for API loginUser, the true test objectives met includes {A, B, C}, and the false objectives met includes {A, B, C, D}, so that the multiple-condition coverage = $\frac{\text{objectives met}}{total\ objectives}$ = $\frac{3+4}{8}$ $\times 100\% = 87.5\%$.

3. For the API updateDetails, all the conditions are listed below (labelled from A to C).

A: if (userIDs.get(sessionID) == null)
B: else if (!Arrays.asList(VALID_URL_PROTOCOLS).contains(url.getProtocol()))
C: if (urlUsername == null || urlPassword == null)

for C there are four conditions. They are <C1,(TRUE,TRUE)><C2,(TRUE,FALSE)><C3,(FALSE,TRUE)><C4,(FALSE,FALSE)>. For others, they each has two different conditions (true or false), so that there are 2 ×2 + 4 = 8 conditions in total.

Then, the multiple-condition coverage score of API updateDetails for the partitioning test is shown in Table 5.3a and that for the boundary value test is shown in Table 5.3b.

Table 5.3a. Partitioning Test Coverage Score for updateDetails

EC	TRUE	FALSE
1	B	A
2	C1	A B

3	C1	A B
4		A B C4
5	A	
6	N/A	N/A
7	N/A	N/A
8	N/A	N/A

Table 5.3b. Boundary Value Test Coverage Score for updateDetails

EC	TRUE	FALSE
1.1	B	A
2.1	C2	A B
3.1	C3	A B
5.1	A	
5.2		A B C4
6.2	N/A	N/A
7.2	N/A	N/A
8.2	N/A	N/A

After running all test cases from the partitioning test for API updateDetails, the true test objectives met includes {A, B, C1}, and the false objectives met includes {A, B, C4}, so that the multiple-condition coverage = $\frac{\text{objectives met}}{total\ objectives}$ =

$\frac{3+3}{8} \times 100\% = 75\%.$

While for the boundary value test for API updateDetails, the true test objectives met includes {A, B, C2, C3}, and the false objectives met includes {A, B, C4}, so that the multiple-condition coverage = $\frac{\text{objectives met}}{total\ objectives}$ = $\frac{4+3}{8} \times 100\% = 87.5\%.$

4. For the API retrieveDetails, all the conditions are listed below (labelled from A to D).

A: if (userIDs.get(sessionID) == null)

B: else if (!Arrays.asList(VALID_URL_PROTOCOLS).contains(url.getProtocol()))

C: if (details.get(passbookUsername) == null)

D: if (pair == null)

Every situation has two different conditions (true or false), so that there are 2 ×4 = 8 conditions in total.

Then, the multiple-condition coverage score of API retrieveDetails for the partitioning test is shown in Table 5.4a and that for the boundary value test is shown in Table 5.4b.

Table 5.4a. Partitioning Test Coverage Score for retrieveDetails

EC	TRUE	FALSE
1	B	A
2	D	A B C
3		A B C D
4	C	A B

5	A	
6	N/A	N/A

Table 5.4b. Boundary Value Test Coverage Score for retrieveDetails

EC	TRUE	FALSE
1.1	B	A
2.1	D	A B C
4.1	C	A B
5.1	A	
5.2		A B C D
6.2	N/A	N/A

After running all test cases from the partitioning test for API retrieveDetails, the true test objectives met includes {A, B, C, D}, and the false objectives met includes {A, B, C, D}, so that the multiple-condition coverage = $\frac{\text{objectives met}}{total\ objectives}$ = $\frac{4+4}{8} \times 100\% = 100\%$.

While for the boundary value test for API retrieveDetails, the true test objectives met includes {A, B, C, D}, and the false objectives met includes {A, B, C, D}, so that the multiple-condition coverage = $\frac{\text{objectives met}}{total\ objectives}$ = $\frac{4+4}{8} \times 100\% = 100\%$.

Task 7

For the two kinds of testing methods partitioning test and boundary value test, the latter is considered to be an improved version of the former. While according to the results in task 5, the partitioning test reaches the performance in most cases. For instance, the test cases for API addUser, updateDetails and retrieveDetails. While for the API loginUser, it reaches the better performance in boundary value test. Therefore, based on the results, it should be found that both the partitioning test and boundary value test have the same input domain. The difference is boundary value test has already suitable test cases, so it may get a better performance.