SWEN90006 Software Testing and Reliability

Assignment 1 Semester 2 – 2019

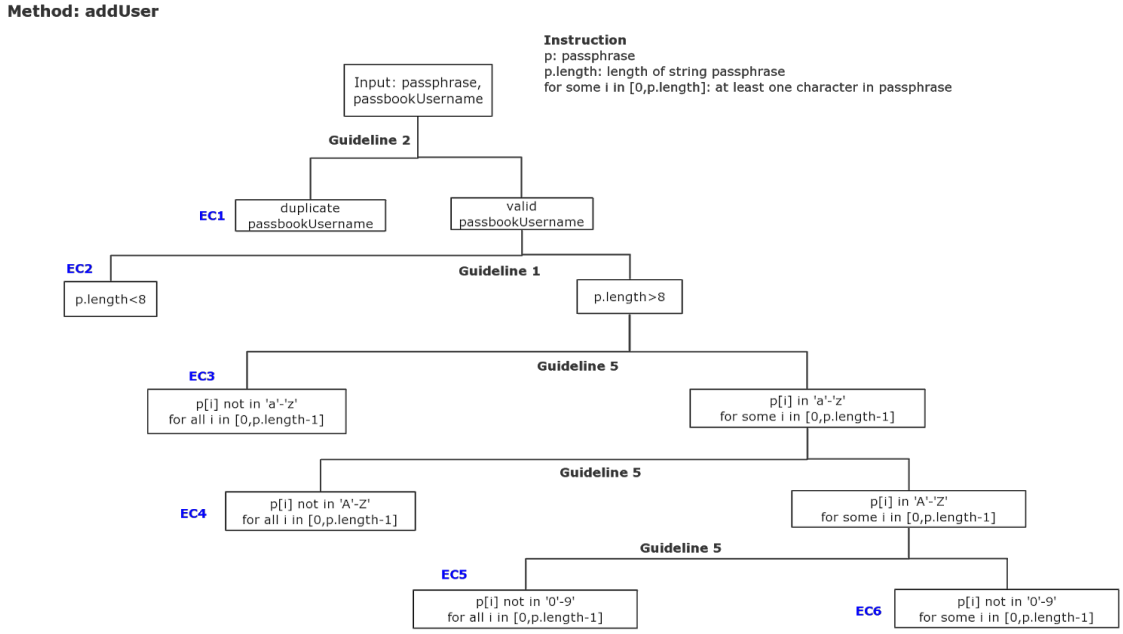
NAME: WENXIAN SONG
STUDENT ID: 935057

Task 1

1. `addUser` method
   First examining the requirement in the specification related to variables.
   Input domain:
   String `passbookUsername` and String `passphrase` are strings.
   Input condition:
   a. String `passbookUsername` is new string that is not used before.
   b. String `passphrase` is

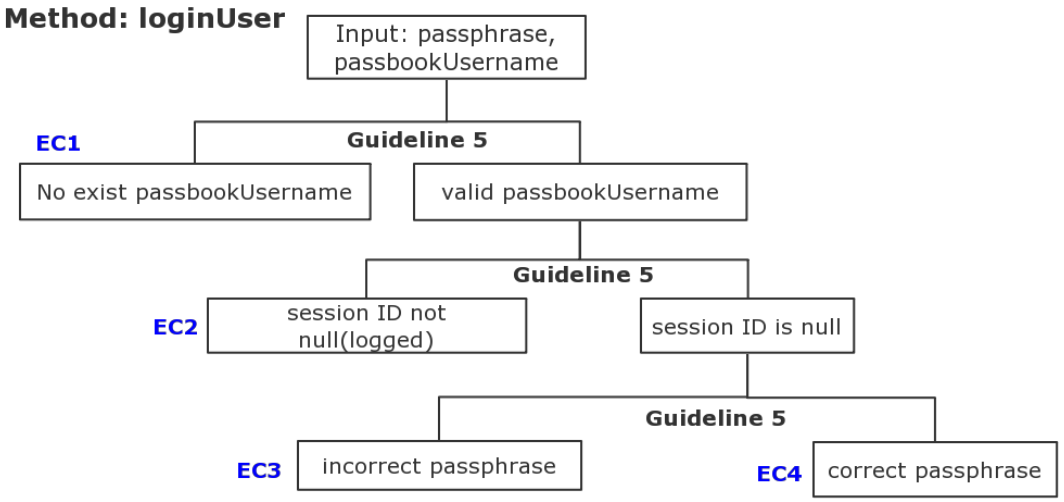
$$\{p \mid p.length \geq 8 \wedge for\ character\ a\ in\ the\ domain\ of\ p,\ a \in [0,9] \wedge for\ character\ b\ in\ the\ domain\ of\ p,\ b \in [a,z] \wedge for\ character\ c\ in\ the\ domain\ of\ p,\ c \in [A,Z]\}$$

   Assumption: input strings are not null, and the number of parameters is correct since this error will be tested in compile time.
   I think ECs cover all the input domain and without overlap.
   In this case, we first split equivalent class (EC) for String `passbookUsername`, one is a new string that never stored before and the other is an invalid class that is duplicated. Then we split the valid class by the length of String `passphrase`, one is length < 8 (EC2) and the other is length >=8; then we examine the lowercase requirement for input satisfied length requirement, and split it into two classes, one is no lowercase character in the input string (EC3), the other is containing at least one lowercase letter in the input string;similarly, based on the uppercase requirement and digit requirement, we can also split EC, details are shown in the figure.



2. `loginUser` method
   First examining the input domain of variables and parameters:
   String `passbookUsername` and `passphrase` are strings.
   Input condition is shown below

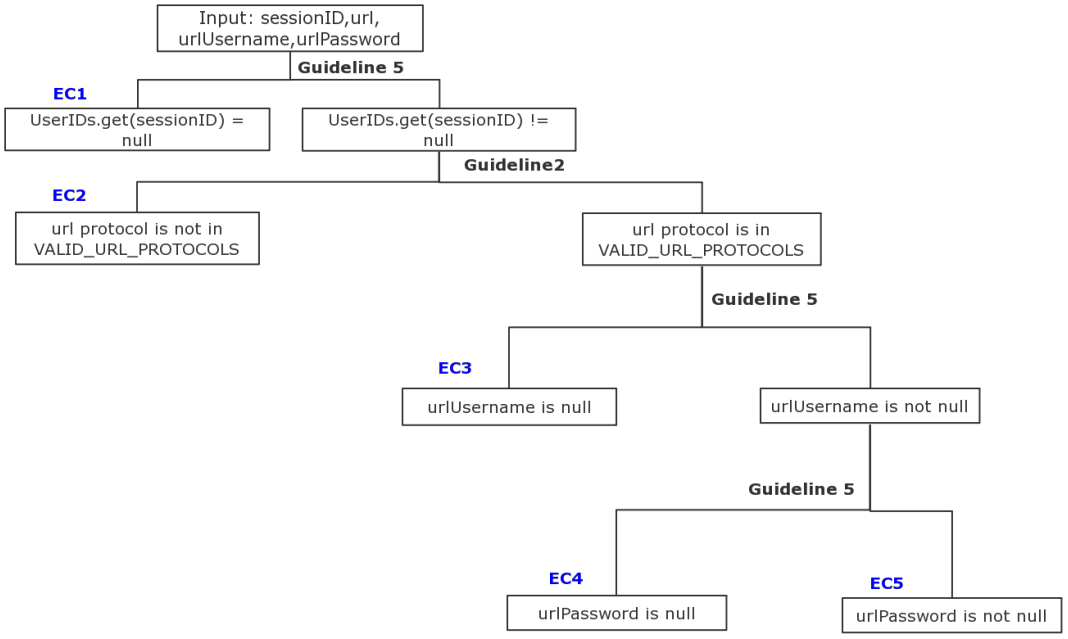
a. String `passbookUsername` should be an existent username stored in the Map `passphrases`
b. String `passbookUsername` should not have corresponding session stored in the Map `sessionIDs`
c. String `passphrase` should match the corresponding `passbookUsername`



Assumption: input strings are not null, and the number of parameters is correct since this error will be tested in compile time.
I think ECs cover all the input domain and without overlap.
According to the input condition above, first we check `passbookUsername`, and split the input into two cases, one is valid username, one is not existent username EC1. Then for the valid `passbookUsername`, we check the session of this `passbookUsername` and split it into two cases, one is logged, and the other is non-record user; then we check the `passphrase` to see it match or not.

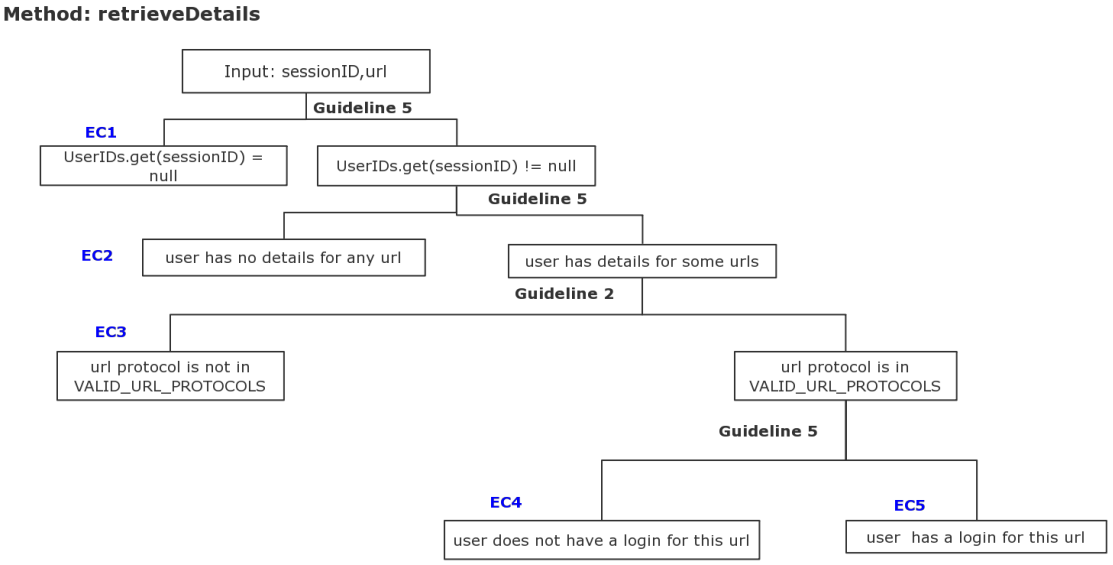
3. `updateDetails` method
First examing the input domain of variables and parameters:
Integer `sessionID`, URL `url`, String `urlUsername` and String `urlPassword`
Input condition is shown below
a. String `sessionID` should belong to an existent user stored in the Map `userIDs`
b. URL `url` should be a valid protocol in the set {HTTP,HTTPS}
c. The `urlUsername` and `urlPassword` should both not null

Method: updateDetails



Assumption: input url is a valid object and not null, integer sess
ion is not null.
I think ECs cover all the input domain and without overlap.
According to the input condition, first we check whether the `sessionID` belongs to a specific user, and split the input into two cases(EC1 is invalid). Then we examine the input url protocol type into two sets(EC2 is invalid). Then for the valid protocols, we have to check whether the `urlUsername` and `urlPassword` is null or not that in EC3, EC4 and EC5.

4. `retrieveDetails` method
   First examing the input domain of variables and parameters:
   Integer `sessionID`, URL `url`
   Input condition is shown below
   a. String `sessionID` should belong to an existent user stored in the Map `userIDs`
   b. URL `url` should be a valid protocol in the set {HTTP,HTTPS}
   c. The user holding this `sessionID` should has detailed information with urls.
   d. The detailed information should contain this input `url` information.



Assumption: input url is a valid object and not null, integer session is not null.
I think ECs cover all the input domain and without overlap.
According to the input condition, first we check whether the `sessionID` belongs to a specific user(EC1), and split the input into two cases. Then we check this valid user's detailed information of Password Table(EC2) and split it into two cases. Then we examine the input url protocol type into two sets(EC3 is invalid). Then for the valid protocols, we have to check whether this user has a login for this url that in EC4 and EC5.

Task 2

Detailed code in the partitioningTests.java

Task 3

1. `addUser` method

For EC1, the boundary is "no duplicate username in `passbookUsername`", then we select one valid test case and one invalid.
For EC2, the boundary is "length of `passphrase` not less than 8 ", then the on point is length = 8, and two off points one is length <8, one is length >8.
For the parameter `passphrase`, the boundary is "there are at least one lowercase letter, uppercase letter and digit ", so the on points should be the number of lowercase = 1, the number of uppercase = 1, the number of digit = 1. And the off points should be the number of lowercase = 0, the number of lowercase >1, the number of uppercase = 0, the number of uppercase >1, the number of digit = 0, the number of digit >1.
From ascii table we know that symbol "/" is below "0", and ":"is above "9", "@" and "[" is the boundary of "A" - "Z", "`" and "{" is the boundary of "a" - "z".
Assuming that there is one current registered username is "test".

EC	boundary	op point	off point
EC1	duplicate username	test, 123qweOP	abc, 123qweOP
EC2	length <8	abc, 123qweOP	abc,123qweO
EC3	no characrter in `passphrase` is lowercase letter	abc, 123`{EOP abc, AAAAAAAA abc, 00000000	abc, 123qWEOP
EC4	no characrter in `passphrase` is uppercase letter	abc, 123qwe@[	abc, 123qweoP
EC5	no character in `passphrase` is digit in '0'-'9'	abc, //:qweOP	abc, 1qweasOP
EC6	`passphrase` contains digit, uppercase, lowercase	abc, 123qweOP abc, aaa000ZZ abc, zzzAA999	abc, /:@[]`{} abc, aaaaaaaa

2. `loginUser` method

For EC1, the boundary is "`passbookUsername` not exist", on point is a invalid `passbookUsername`, and off point is a registered `passbookUsername`.
For EC2, the boundary is "session ID not null", on point is a already logged user, and the off point is a new user without sessionID.
For EC3, for the no session user, the boundary is "incorrect `passphrase`", on point is a wrong `passphrase`, and the off point is a correct `passphrase`.
For EC4, the boundary is "a valid `passbookUsername` that never log in and with correct `passphrase`", the on point is a correct new pair, the off point is randomly in EC1,EC2,EC3.
Assuming that there is one current new registered pair is "abc, 123qweOP", a logged username is "test".

EC	boundary	op point	off point
EC1	`passbookUsername` is not exist	xxx, 123qweOP	abc, 123qweOP
EC2	session ID not null	test, 123qweOP	abc, 123qweOP
EC3	incorrect `passphrase`	abc, qqqqqqqq	abc, 123qweOP
EC4	a valid `passbookUsername` that not login with correct `passphrase`	abc, 123qweOP	test, 123qweOP

3. `updateDetails` method

For EC1, the boundary is "no user has this input `sessionID` ", the on point is an invalid `sessionID`, and the off point is a valid one.
For EC2, the boundary is "protocol is invalid", here we select one on point that not a valid one, and select one off point that in the set of valid set {http,https}.
For EC3, the boundary is "`urlUsername` is null", and we also use Guideline 3 to select an invalid `urlUsername` and one valid `urlUsername`.

For EC4 and EC5, when the `urlUsername` is valid,the boundary is "`urlPassword` is null", and we also use Guideline 3 to select an invalid `urlPassword` and one valid `urlPassword`.
Assuming that user "abc" has sessionID=123456, with `url`= "*http://www.processon.com/diagrams*", urlUsername `user` = "xxx", password `pwd`="123qweasd".
we select an invalid url, `invalid_url` = ***hhhh://www.processon.com/diagrams***

EC	boundary	op point	off point
EC1	`sessionID` is invalid	111111,`url,user,pwd`	123456,`url,user,pwd`
EC2	protocol is not in `VALID_URL_PROTOCOLS`	123456, `invalid_url,user,pwd`	123456, `url,user,pwd`
EC3	null urlUsername	123456, `url,NULL,pwd`	123456, `url,user,pwd`
EC4	null urlPassword	123456, `url,user,NULL`	123456, `url,user,pwd`
EC5	not null urlPassword and urlUsername	123456, `url,user,pwd`	123456, `url,NULL,NULL`

4. `retrieveDetails` method

For EC1, the boundary is "no user has this input `sessionID` ", the on point is an invalid `sessionID`, and the off point is a valid one.
For EC2, the boundary is "user has no url information", the on point is a valid session user with empty details for any url. The off point is a valid session user with a corresponding details. Due to the implementation of code, we cannot check this condition(the statement will always return true).
For EC3, the boundary is "protocol is invalid", here we select one on point that not a valid one, and select one off point that in the set of valid set {http,https}. For EC4, the boundary is "detailed information does not contain this url information", the on point is a not logged record of this url, off point is a specific url information of `url`.
Assuming that user "abc" has sessionID=123456, with `url`= "*https://www.processon.com/diagrams*",
And user "test" has sessionID = 101010, but no corresponding url information.
we select an invalid url, `invalid_url` = ***hhhh://www.processon.com/diagrams***
and also an HTTPS url that abc never loggin, `new_url` = https://scholar.google.com.au

EC	boundary	op point	off point
EC1	`sessionID` is invalid	111111,`url`	123456,`url`
EC2	user has no details for any url(untestable)	101010, `url`	123456, `url`
EC3	protocol is not in `VALID_URL_PROTOCOLS`	123456, `invalid_url`	123456, `url`
EC4	user does not log in for this url	123456, `new_url`	123456, `url`
EC5	user has a log in for this url	123456, `url_`	123456, `new_url`

There are overlaps between different test cases, so for the implementation we can reuse test cases to test the boundary.

Task 4

Detailed code in the boundaryTests.java

Task 5

1. `addUser` method

No.	Branch code	Permutation	met test cases	missed test cases	
				partitioning	boundary
C1	`passphrases.containsKey(passbookUsername)`	{T}{F}	EC1,EC2	0	0
C2	`passphrase.length() < MINIMUM_PASSPHRASE_LENGTH`	{T}{F}	EC2,EC3	0	0
C3	`i < passphrase.length()`	{T}{F}	EC4	0	0
C4	`if ('a' <= passphrase.charAt(i) && passphrase.charAt(i) <= 'z')`	{T,T} {T,F} {F,T} {F,F}	EC4 EC3 EC3 -(untestable)	1	1
C5	`if ('A' <= passphrase.charAt(i) && passphrase.charAt(i) <= 'Z')`	{T,T} {T,F} {F,T} {F,F}	EC5 EC4 EC4 -(untestable)	1	1
C6	`if ('0' <= passphrase.charAt(i) && passphrase.charAt(i) <= '9')`	{T,T} {T,F} {F,T} {F,F}	EC6 EC5 EC5 -(untestable)	1	1
C4& C5& C6	`if (!containsLowerCase || !containsUpperCase || !containsNumber)`	{T,T,T} {T,F,T} {T,F,F} {T,T,F} {F,T,T} {F,T,F} {F,F,T} {F,F,F}	- (boundary) EC3 (boundary) (boundary) EC4 EC5 EC6	4	1

Total number of test cases = 26
Missing test case of partitioning = 7
Missing test case of boundary = 1

There are three cases that is untestable, for example a character that satisfy $(c \leq 'a') \wedge (c \geq 'z')$ is not exist logically.
The score of equivalence partitioning = 19/26 = 73.1%
The score of boundary analysis = 22/26 = 84.6%

2. `loginUser` method

No.	Branch code	Permutation	met test cases	missed test cases	
				partitioning	boundary
C1	`if (!passphrases.containsKey(passbookUsername)))`	{T} {F}	EC1 EC2	0	0
C2	`if (sessionIDs.get(passbookUsername) != null)`	{T} {F}	EC2 EC3	0	0
C3	`if (!passphrases.get(passbookUsername).equals(passphrase))`	{T} {F}	EC3 EC4	0	0
C4	`while (userIDs.containsKey(sessionID))`	{T} {F}	-(untestable) - EC4	1	1

Total number of test cases = 8
Missing test cases of partitioning = 1
Missing test cases of boundary = 1
There is one test case that is unreachable, which means is difficult to determine the true condition of the while loop, so we do not consider this case as a possible test objects.
The score of equivalence partitioning = 7/8 = 87.5%
The score of boundary analysis = 7/8 = 87.5%

3. `updateDetails` method

No.	Branch code	Permutation	met test cases	missed test cases	
				partitioning	boundary
C1	`if (passbookUsername == null)`	{T} {F}	EC1 EC2	0	0
C2	`if (!Arrays.asList(VALID_URL_PROTOCOLS).contains(url.getProtocol()))`	{T} {F}	EC2 EC3	0	0

C3	`if (urlUsername == null || urlPassword == null)`	{T,T} {T,F} {F,T} {F,F}	-(boundary) EC3 EC4 EC5	1	0

Total number of test cases = 8
Missing test cases of partitioning = 1
Missing test cases of boundary = 0
The score of equivalence partitioning = 7/8 = 87.5%
The score of boundary analysis = 100%

4. `retrieveDetails` method

No.	Branch code	Permutation	met test cases	missed test cases	
				partitioning	boundary
C1	`if (passbookUsername == null)`	{T} {F}	EC1 EC5	0	0
C2	`if (!Arrays.asList(VALID_URL_PROTOCOLS).contains(url.getProtocol()))`	{T} {F}	EC3 EC4	0	0
C3	`if (pt == null)`	{T} {F}	- (always false)	1	1
C4	`if (pair == null)`	{T} {F}	EC4 EC5	0	0

Total number of test cases = 8
Missing test cases of partitioning = 1
Missing test cases of boundary = 1
Because there is one test case that is untestable logically, each user has PassTable. In this case, C3 is always false.
So finally the score of equivalence partitioning = 7/8 = 87.5%
The score of boundary analysis = ⅞ = 87.5%

Task 6

Detailed code in the boundaryTests.java

Task 7

1. The coverage of valid input/output domain
   Partitioning test and boundary test are both able to cover the valid input domain and output domain. Because the boundary is derived from the partition of equivalence classes and extended from each EC, they cover the same input domain and output doman.
2. The coverage score
   From all the methods, the score of boundary test is higher than or equal to partitioning test. Because the boundary test cases consider more possibilities and try to add more test cases to both on points and off points. For example, change the condition of passphrase validation, from `'a' <= passphrase.charAt(i)"` to `'a' < passphrase.charAt(i)",` the correct region is smaller than original one. And boundary test can find off points that cannot be found by partitioning test, like the symbol '`' which ascii value less than 'a'.
3. Mutants
   Most of mutants are killed by boundary tests actually, because most effective mutants are derived from the boundary changes that are common mistakes in programming. In contrast, partitioning test cases are selected randomly in the ECs, which may be not able to cover all the boundary in the program.
4. Summary
   In a word, these two types of method are effective in testing and boundary test is a kind of complementing of partitioning test that extend the test cases for marginal value. The overall performance is better on boundary test.