2019

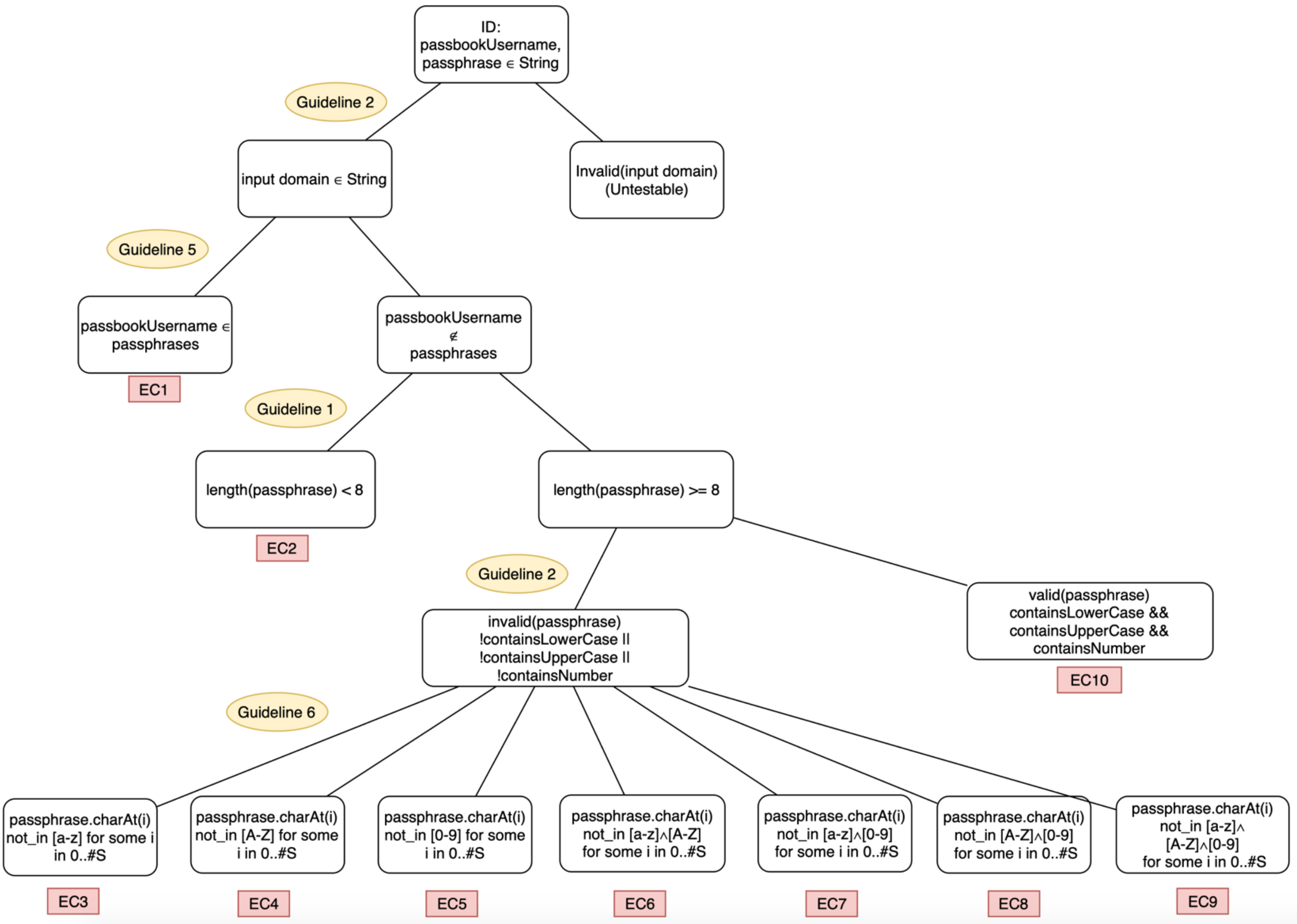
SWEN90006: Software Testing and Reliability

ASSIGNMENT 1

Student Name: ZIQI JIA
Student Number: 693241

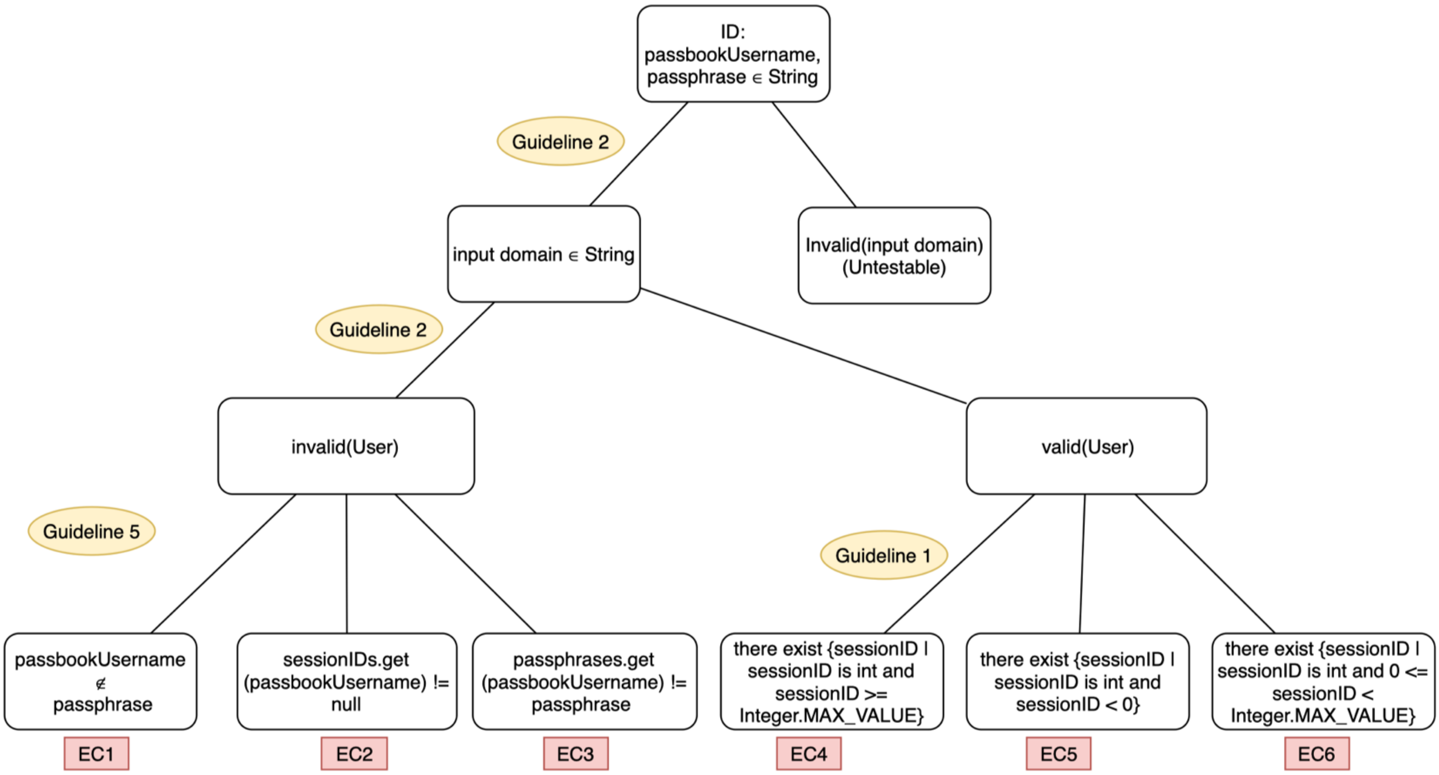
Task 1 Equivalence Partitioning

1. addUser() Template Tree



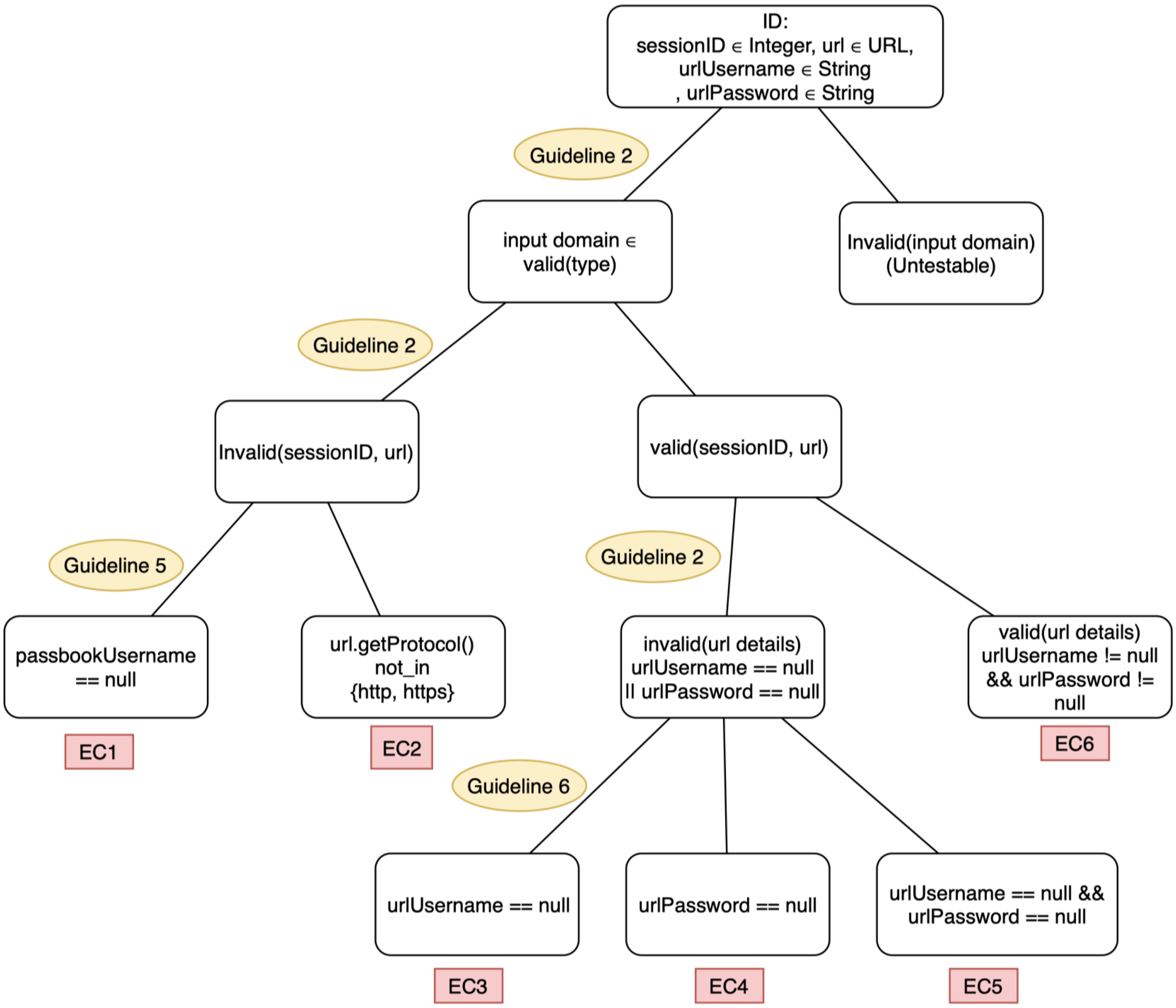
Assumed that the data type of input space has to be correct. The valid input domain can be divided into valid passbookUsername and invalid passbookUsername. For the invalid passbookUsername, the EC considers the exception “DuplicateUserException” through EC1. The valid username will be then decomposed into passphrase of length less than or greater and equal to 8. For passphrase length less than 8, the exception “WeakPassphraseException” will be expected. For passphrase length greater and equal to 8, the valid passphrase must contain lowercase, uppercase and number at the same time, otherwise will through “WeakPassphraseException” exception (EC3-EC9).

2. loginUser() Template Tree



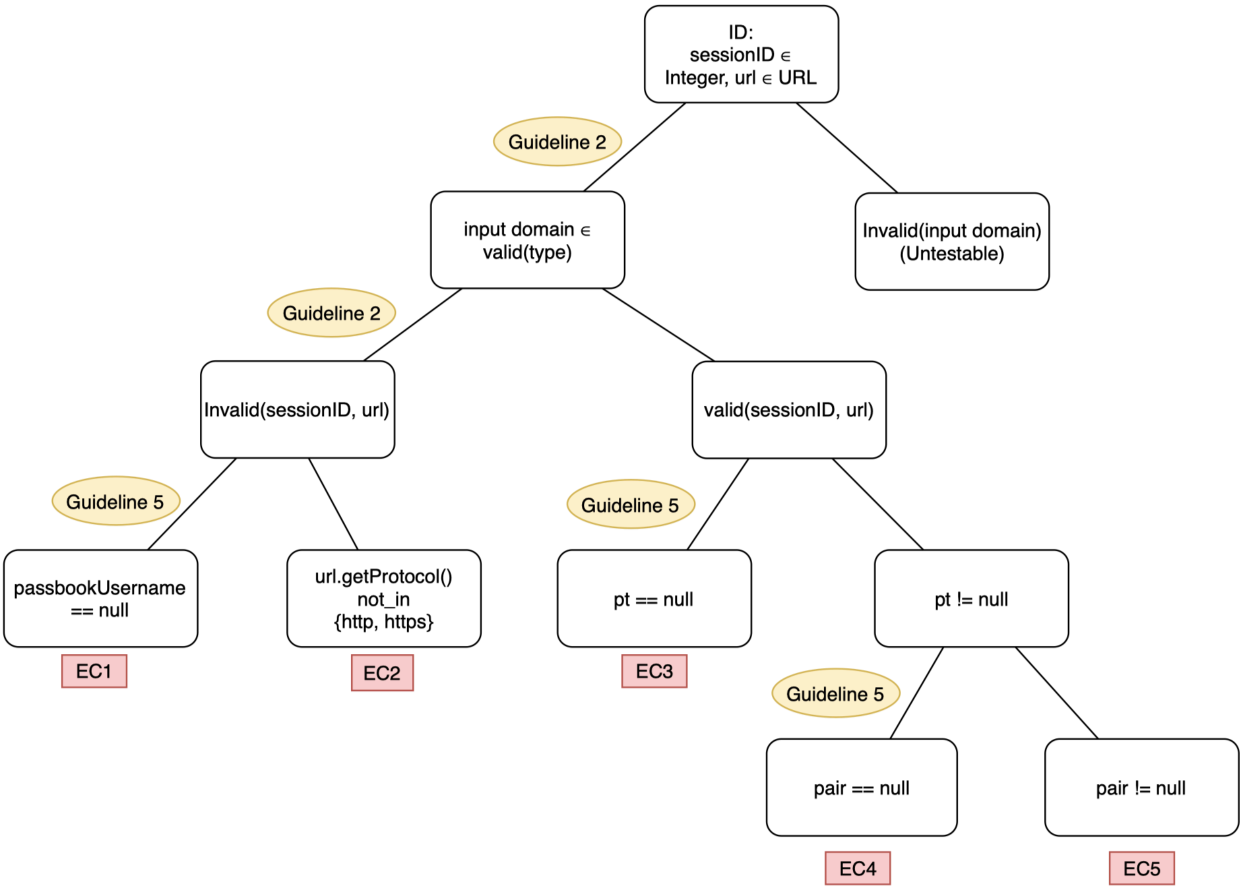
Again, it is assumed that the input program will be String type. The valid input domain is divided into valid user and invalid user. For the invalid user, the EC considers the three exceptions “NoSuchUserException”, “AlreadyLoggedInException”, and “IncorrectPassphraseException” through EC1, EC2, EC3. For the valid user, the EC considers the valid range of sessionID, that is, between 0 to integer’s max value. Otherwise the sessionID will be invalid.

3. updateDetails() Template Tree



Assume the data type of input space have to be correct, the valid input domain is decomposed into valid sessionID and url and invalid sessionID and url. For the invalid sessionID and url, the EC considers the exceptions “InvalidSessionIDException” and “MalformedURLException” through EC1 and EC2. The valid sessionID and url can be further branched out into valid url details and invalid url details. The valid url details must contain not null urlUsername and not null urlPassword, otherwise the updating details will be failed and url detail will be removed from the password table. the EC considers the invalid url details with exception “NoSuchURLException” through EC3, EC4, EC5.

4. retrieveDetails() Template Tree



Assumed the input program will be correct data type. The valid input domain is decomposed into valid sessionID and url and invalid sessionID and url. For the invalid sessionID and url, the ECs are exectly same as the updateDetails(). the EC considers the exceptions “InvalidSessionIDException” and “MalformedURLException” through EC1 and EC2. For valid sessionID and url, the EC considers the valid url details through EC5 and invalid url details through the exception “NoSuchURLException” (EC3 and EC4).

Discussion:

Overall, because it is safe to assume that the input program contains the correct data types, we can decompose only valid input domain. The new node resulting from decomposition will be further divided into in a similar manner, taking into account possible values, and following the provided equivalence division guidelines. Therefore, the equivalence class (EC) covers the input space sufficiently for each method.

Task 3 Boundary value analysis

Boundary value analysis (BVA) for the ECs will be conducted. The first division of the ECs is into invalid and valid domains. The incorrect data type of input program is invalid because it assumes that the data type of the input program is always correct, so this is an untestable domain. For the rest testable domain, the unbounded equivalence classes uses the original test case from the equivalence partition. Otherwise, select the points on and off in each bounded equivalence class. If two or more equivalence class generates the same on or off points, only one will be selected because the test cases are the same.

1. addUser() Boundary Value Analysis

EC	Boundaries	On point	Off point
1	{ passbookUsername|passbookUsername ∈ passphrases}	passbookUsername ∈ passphrases is **true**	passbookUsername ∈ passphrases is **false** passbookUsername = null
2	{ passbookUsername, passphrase | length(passphrase) < 8 ∧ passbookUsername ∉ passphrases }	length(passphrase) = 8	length(passphrase) = 7 length(passphrase) = 0
3	{ passphrase | passphrase.charAt(i) not_in [a-z] ∧ length(passphrase) >= 8}	Passphrase = aaaaaaaa Passphrase = zzzzzzzz (length(passphrase) = 8)	Based on ascii chart: Passphrase = ``ABC0123 Passphrase = {{ZYX9876 (length(passphrase) = 9) As ‘`’ = ‘a’-1, ‘{‘ = ‘z’+1.
4	{ passphrase | passphrase.charAt(i) not_in [A-Z] ∧ length(passphrase) >= 8}	Passphrase = AAAAAAAA Passphrase = ZZZZZZZZ (length(passphrase) = 8)	Based on ascii chart: Passphrase = @@abc0123 Passphrase = [[zyx9876 (length(passphrase) = 9) As ‘@’ = ‘A’-1, ‘[‘ = ‘Z’+1.
5	{ passphrase | passphrase.charAt(i) not_in [0-9] ∧ length(passphrase) >= 8}	Passphrase = 0000000	Based on ascii chart: Passphrase = ///abcABC

		Passphrase = 99999999 (length(passphrase) = 8)	Passphrase = :::xyzXYZ (length(passphrase) = 9) As ‘/’ = ‘0’-1, ‘:‘ = ‘9’+1.
6	{ passphrase | passphrase.charAt(i) not_in ([a-z]∧[A-Z]) ∧ length(passphrase) >= 8}	Passphrase = aaaaAAAA Passphrase = zzzzZZZZ (length(passphrase) = 8)	Have identical tests with EC5’s on point.
7	{ passphrase | passphrase.charAt(i) not_in ([a-z]∧[0-9]) ∧ length(passphrase) >= 8}	Passphrase = aaaa0000 Passphrase = zzzz9999 (length(passphrase) = 8)	Have identical tests with EC4’s on point.
8	{ passphrase | passphrase.charAt(i) not_in ([0-9]∧[A-Z]) ∧ length(passphrase) >= 8}	Passphrase = AAAA0000 Passphrase = ZZZZ9999 (length(passphrase) = 8)	Have identical tests with EC3’s on point.
9	{ passphrase | passphrase.charAt(i) not_in ([a-z]∧[A-Z]∧[0-9]) ∧ length(passphrase) >= 8}	Passphrase = aAAAAAA0 Passphrase = zZZZZZZ9 (length(passphrase) = 8)	Passphrase = {{{{{{{{{ Passphrase = ///////// As ‘/’ = ‘0’-1, ‘{‘ = ‘z’+1.
10	{ passphrase | passphrase.charAt(i) in ([a-z]∧[A-Z]∧[0-9]) ∧ length(passphrase) >= 8}	Have identical tests with EC9’on points.	Passphrase = bBBBBBBB1 Passphrase = yYYYYYYY8 (length(passphrase) = 9)

2. loginUser() Boundary Value Analysis

EC	Boundaries	On point	Off point
1	{ passbookUsername | passbookUsername ∉ passphrases}	passbookUsername ∉ passphrases is **true**	passbookUsername ∉ passphrases is **false**
2	{ passbookUsername | passbookUsername ∉ sessionIDs }	passbookUsername ∉ sessionIDs is **true**	passbookUsername ∉ sessionIDs is **false**
3	{ passbookUsername, passphrase | passphrases.get(passbookUsername) != passphrase}	passphrases.get (passbookUsername) != passphrase is **true**	passphrases.get (passbookUsername) != passphrase is **false**
4	{sessionID | sessionID is int and sessionID >= Integer.MAX_VALUE} Since sessionID is a random number, the specific number for sessionID is untestable, can only test if the sesseionID is within the range.	sessionID = Integer.MAX_VALUE (untestable)	sessionID = Integer.MAX_VALUE + 1 (untestable)
5	{sessionID | sessionID is int and sessionID < 0} Since sessionID is a random number, the specific number for sessionID is untestable, can only test if the sesseionID is within the range.	sessionID = 0 (untestable)	sessionID = -1 (untestable)
6	{sessionID | sessionID is int and 0 <= sessionID < Integer.MAX_VALUE} Since sessionID is a random number, the specific number for sessionID is untestable, can only test if the sesseionID is within the range.	sessionID = 0 (untestable) sessionID = Integer.MAX_VALUE (untestable)	sessionID = 1 (untestable) sessionID = Integer.MAX_VALUE-1 (untestable)

3. updateDetails() Boundary Value Analysis

EC	Boundaries	On point	Off point
1	{ sessionIDs | passbookUsername == null}	passbookUsername == null	passbookUsername != null
2	{ url | url.getProtocol() not_in {http, https}}	url.getProtocol() = http url.getProtocol() = https	url.getProtocol() = httpp url = http
3	{ urlUsername | urlUsername == null }	urlUsername == null	urlUsername != null
4	{ urlPassword | urlPassword == null}	urlPassword == null	urlPassword != null
5	{ urlUsername, urlPassword | urlUsername == null && urlPassword == null}	urlUsername == null && urlPassword == null	urlUsername != null && urlPassword != null
6	{ urlUsername, urlPassword | urlUsername != null && urlPassword != null}	Have identical tests with EC5'off points.	Have identical tests with EC5'on points.

4. retrieveDetails() Boundary Value Analysis

EC	Boundaries	On point	Off point
1	{ sessionIDs | passbookUsername == null}	passbookUsername == null	passbookUsername != null
2	{ url| url.getProtocol() not_in {http, https}}	Have identical tests with EC2 of updateDetails().	url.getProtocol() = httpp
3	{ sessionIDs , url |pt == null}	pt == null	pt != null
4	{ sessionIDs , url | pt != null ∧ pair == null}	pt != null ∧ pair == null	pt == null ∧ pair != null
5	{ sessionIDs , url | pt != null ∧ pair != null}	pt != null ∧ pair != null	pt == null ∧ pair == null

Task 5 multiple-condition coverage

1. addUser Method

in addUser Method, there are:
2 if statement containing a single condition = 2 × 2^1 = 4
3 if statement containing two conditions = 3 × 2^2 = 12
1 if statement containing three conditions = 1 × 2^3 = 8
1 for statement containing a single condition = 1 × 2^1 = 2
So, there are 4 + 12 + 8 + 2 = 26 conditions possible in total.

<table>
<tr><th></th><th>Condition</th><th>Possible output</th><th>Objective</th></tr>
<tr><td rowspan="2">1</td><td rowspan="2">if (passphrases.containsKey(passbookUsername))</td><td>true</td><td>1</td></tr>
<tr><td>false</td><td>2</td></tr>
<tr><td rowspan="2">2</td><td rowspan="2">if (passphrase.length() < MINIMUM_PASSPHRASE_LENGTH)</td><td>true</td><td>3</td></tr>
<tr><td>false</td><td>4</td></tr>
<tr><td rowspan="4">3</td><td rowspan="4">if ('a' <= passphrase.charAt(i) && passphrase.charAt(i) <= 'z')</td><td>true false</td><td>5</td></tr>
<tr><td>false false</td><td>6</td></tr>
<tr><td>false true</td><td>7</td></tr>
<tr><td>true true</td><td>8</td></tr>
<tr><td rowspan="4">4</td><td rowspan="4">else if ('A' <= passphrase.charAt(i) && passphrase.charAt(i) <= 'Z')</td><td>true false</td><td>9</td></tr>
<tr><td>false false</td><td>10</td></tr>
<tr><td>false true</td><td>11</td></tr>
<tr><td>true true</td><td>12</td></tr>
<tr><td rowspan="4">5</td><td rowspan="4">else if ('0' <= passphrase.charAt(i) && passphrase.charAt(i) <= '9')</td><td>true false</td><td>13</td></tr>
<tr><td>false false</td><td>14</td></tr>
<tr><td>false true</td><td>15</td></tr>
<tr><td>true true</td><td>16</td></tr>
<tr><td rowspan="8">6</td><td rowspan="8">if (!containsLowerCase || !containsUpperCase || !containsNumber)</td><td>true false true</td><td>17</td></tr>
<tr><td>false false true</td><td>18</td></tr>
<tr><td>false true true</td><td>19</td></tr>
<tr><td>true true true</td><td>20</td></tr>
<tr><td>true false false</td><td>21</td></tr>
<tr><td>false false false</td><td>22</td></tr>
<tr><td>false true false</td><td>23</td></tr>
<tr><td>true true false</td><td>24</td></tr>
<tr><td rowspan="2">7</td><td rowspan="2">for (int i = 0; i < passphrase.length(); i++)</td><td>true</td><td>25</td></tr>
<tr><td>false</td><td>26</td></tr>
</table>

Multiple-condition coverage in equivalence partitioning		
EC	Test Case	Meet with objective
1	pb.addUser(“passbookUsername”, “properPassphrase1”); pb.addUser(“passbookUsername”, “properPassphrase1”);	1
2	pb.addUser(“passbookUsername”, “abCD567”);	2, 3
3	pb.addUser(“passbookUsername”, “AAA11111”);	2, 4, 7, 11, 12, 13, 16, 21, 25, 26
4	pb.addUser(“passbookUsername”, “aaa11111”);	2, 4, 7, 8, 9, 11, 13, 16, 23, 25, 26
5	pb.addUser(“passbookUsername”, “aaaaAAAA”);	2, 4, 7, 8, 9, 12, 13, 18, 25, 26
6	pb.addUser(“passbookUsername”, “11111111”);	2, 4, 7, 11, 16, 24, 25, 26
7	pb.addUser(“passbookUsername”, “AAAAAAAA”);	2, 4, 7, 12, 13, 17, 25, 26
8	pb.addUser(“passbookUsername”, “aaaaaaaa”);	2, 4, 8, 9, 13, 19, 25, 26
9	pb.addUser(“passbookUsername”, “``{{@[[/:”);	2, 4, 5, 7, 9, 11, 13, 15, 20, 25, 26
10	pb.addUser(“passbookUsername”, “ properPassphrase1”);	2, 4, 7, 8, 9, 11, 12, 13, 16, 22, 25, 26

With equivalence partitioning, 3 conditions are not met, that is, objective 6, 10, 14. Condition 6, 10, 14 are logically not possible based on ascii chart, as we could not select the passphrase that satisfy both false ouput at the same time for the if statement that contain two conditions in this method, because if one condition is false, the other one condition must be true.

Hence, multiple-condition coverage result for equivalence partitioning = 23/26 = 88.46%

Multiple-condition coverage in boundary analysis		
EC	Test Case	Meet with objective
1a	pb.addUser("passbookUsername", " properPassphrase1"); pb.addUser("passbookUsername", " properPassphrase1");	1
1b	pb.addUser(null , "properPassphrase1");	2, 4, 7, 8, 9, 11, 12, 13, 16, 22, 25, 26
2a	pb.addUser("passbookUsername", "abcdEFG8");	2, 4, 7, 8, 9, 11, 12, 13, 16, 22, 25, 26
2b	pb.addUser("passbookUsername", "abCD567");	2, 3
2c	pb.addUser("passbookUsername", null);	2, 3
3a	pb.addUser("passbookUsername", "aaaaaaaa");	2, 4, 8, 9, 13, 19, 25, 26
3b	pb.addUser("passbookUsername", "zzzzzzzz");	2, 4, 8, 9, 13, 19, 25, 26

3c	pb.addUser("passbookUsername", "``ABC0123");	2, 4, 7, 9, 11, 12, 13, 16, 21, 25, 26
3d	pb.addUser("passbookUsername", "{{ZYX9876");	2, 4, 5, 7, 9, 11, 12, 13, 16, 21, 25, 26
4a	pb.addUser("passbookUsername", "AAAAAAAA");	2, 4, 7, 12, 13, 17, 25, 26
4b	pb.addUser("passbookUsername", "ZZZZZZZZ");	2, 4, 7, 12, 13, 17, 25, 26
4c	pb.addUser("passbookUsername", " @@abc0123");	2, 4, 7, 8, 9, 11, 13, 16, 23, 25, 26
4d	pb.addUser("passbookUsername", " [[zyx9876");	2, 4, 7, 8, 9, 11, 13, 16, 23, 25, 26
5a	pb.addUser("passbookUsername", " 00000000");	2, 4, 7, 11, 16, 24, 25, 26
5b	pb.addUser("passbookUsername", " 99999999");	2, 4, 7, 11, 16, 24, 25, 26
5c	pb.addUser("passbookUsername", " ///abcABC ");	2, 4, 7, 8, 9, 11, 12, 13, 15, 18, 25, 26
5d	pb.addUser("passbookUsername", " :::xyzXYZ");	2, 4, 7, 8, 9, 11, 12, 13, 18, 25, 26
6a	pb.addUser("passbookUsername", " aaaaAAAA");	2, 4, 7, 8, 9, 12, 13, 18, 25, 26
6b	pb.addUser("passbookUsername", " zzzzZZZZ");	2, 4, 7, 8, 9, 12, 13, 18, 25, 26
7a	pb.addUser("passbookUsername", " aaaa0000");	2, 4, 7, 8, 9, 11, 13, 16, 23, 25, 26
7b	pb.addUser("passbookUsername", " zzzz9999");	2, 4, 7, 8, 9, 11, 13, 16, 23, 25, 26
8a	pb.addUser("passbookUsername", " AAAA0000");	2, 4, 7, 11, 12, 13, 16, 21, 25, 26
8b	pb.addUser("passbookUsername", " ZZZZ9999");	2, 4, 7, 11, 12, 13, 16, 21, 25, 26
9a	pb.addUser("passbookUsername", " aAAAAAA0");	2, 4, 7, 8, 9, 11, 12, 13, 16, 22, 25, 26
9b	pb.addUser("passbookUsername", "zZZZZZZ9");	2, 4, 7, 8, 9, 11, 12, 13, 16, 22, 25, 26
9c	pb.addUser("passbookUsername", "{{{{{{{{");	2, 4, 5, 9, 13, 20, 25, 26
9d	pb.addUser("passbookUsername", "////////");	2, 4, 7, 11, 15, 20, 25, 26
10a	pb.addUser("passbookUsername", " bBBBBBB1");	2, 4, 7, 8, 9, 11, 12, 13, 16, 22, 25, 26
10b	pb.addUser("passbookUsername", " yYYYYYY8");	2, 4, 7, 8, 9, 11, 12, 13, 16, 22, 25, 26

With boundary analysis, 3 conditions are not met. As mentioned before, the condition 6, 10, 14 are logically not possible based on ascii chart.
Multiple-condition coverage result for boundary partitioning: 23/26 = 88.46%

2. loginUser Method

in loginUser Method, there are:
3 if statement containing a single condition = 3 × 2^1 = 6
1 while statement containing a single condition = 1 × 2^1 = 2
So there are 6 + 2 = 8 conditions possible in total.

	Condition	**Possible output**	**Objective**
1	if (!passphrases.containsKey(passbookUsername))	true	1
		false	2
2	else if (sessionIDs.get(passbookUsername) != null)	true	3
		false	4
3	else if (!passphrases.get(passbookUsername).equals(passphrase))	true	5
		false	6
4	while (userIDs.containsKey(sessionID))	true	7
		false	8

Multiple-condition coverage in equivalence partitioning		
EC	Test Case	Meet with objective
1	pb.loginUser("passbookUsername", "properPassphrase1"); without addUser	1
2	pb.addUser("passbookUsername", "properPassphrase1"); pb.loginUser("passbookUsername", “properPassphrase1");	2, 3
3	pb.loginUser("passbookUsername", " properPassphrase2");	2, 4, 5
4	pb.loginUser("passbookUsername", " properPassphrase1");	2, 4, 6, 8
5	pb.loginUser("passbookUsername", " properPassphrase1");	2, 4, 6, 8
6	pb.loginUser(getRandomString (20), " properPassphrase1");	2, 4, 6, 7, 8

With equivalence partitioning, 0 condition are not met. For test case 6, we generate 1000000 users to be added and login, so we assume it will meet at least once objective 7.
Multiple-condition coverage result for equivalence partitioning: 8/8 = 100%

Multiple-condition coverage in boundary analysis

With boundary analysis, there are no new test cases exist. Since EC1, EC2, and EC3 have the same test cases with equivalence partitioning, and EC4, EC5, EC6’s boundary analysis values are untestable, we could just inherence from the equivalence partitioning test. So, the Multiple-condition coverage result for boundary partitioning will be same as the coverage result for equivalence partitioning.
Multiple-condition coverage result for boundary partitioning: 8/8 = 100%

3. updateDetails Method

In updateDetails Method, there are:
2 if statement containing a single condition = 2 × 2^1 = 4
1 while statement containing two conditions = 1 × 2^2 = 4
So, there are 4 + 4 = 8 conditions possible in total.

	Condition	**Possible output**	**Objective**
1	if (passbookUsername == null)	true	1
		false	2
2	else if (!Arrays.asList(VALID_URL_PROTOCOLS).contains (url.getProtocol()))	true	3
		false	4
3	if (urlUsername == null || urlPassword == null)	true false	5
		false false	6
		false true	7
		true true	8

Multiple-condition coverage in equivalence partitioning		
EC	Test Case	Meet with objective
1	pb.logoutUser(s); pb.updateDetails(s, “http://1234.com”, “urlUsername”, “urlPassword”);	1
2	pb.updateDetails(s, “htttp://1234.com”, “urlUsername”, “urlPassword”);	2, 3
3	pb.updateDetails(s, “http://1234.com”, null, “urlPassword”);	2, 4, 5
4	pb.updateDetails(s, “http://1234.com”, “urlUsername”, null);	2, 4, 7
5	pb.updateDetails(s, “http://1234.com”, null, null);	2, 4, 6
6	pb.updateDetails(s, “http://1234.com”, “urlUsername”, “urlPassword”);	2, 4, 8

With equivalence partitioning, 0 conditions are not met.
Multiple-condition coverage result for equivalence partitioning: 8/8 = 100%

Multiple-condition coverage in boundary analysis		
EC	Test Case	Meet with objective
2a	url.getProtocol() = http url = "http://1234.com";	2, 4, 8
2b	url.getProtocol() = https url = "https://1234.com";	2, 4, 8
2c	url = "httpp://1234.com";	2, 3

2d	url = "http";	2, 3

With boundary analysis, only EC2 have the different test cases with its boundary values, all the other ECs' test cases are the same as equivalence partitioning test. So, the Multiple-condition coverage result for boundary partitioning will be same as the coverage result for equivalence partitioning.
Multiple-condition coverage result for boundary partitioning: 8/8 = 100%

4. retrieveDetails() Method

In retrieveDetails Method, there are:
4 if statement containing a single condition = 4 × 2^1 = 8
So, there are 8 conditions possible in total.

	Condition	Possible output	Objective
1	if (passbookUsername == null)	true	1
		false	2
2	else if (!Arrays.asList(VALID_URL_PROTOCOLS).contains (url.getProtocol()))	true	3
		false	4
3	if (pt == null)	true	5
		false	6
4	if (pair == null)	true	7
		false	8

Multiple-condition coverage in equivalence partitioning		
EC	Test Case	Meet with objective
1	pb.logoutUser(s); pb.retrieveDetails(s, "http://1234.com");	1
2	pb.retrieveDetails(s, "htttp://1234.com");	2, 3
3	pb.retrieveDetails(s, "http://1234.com"); without updateDetail	2, 4, 5
4	pb.retrieveDetails(s, "http://12345.com"); url is different from updateDetail	2, 4, 6, 7
5	pb.retrieveDetails(s, "http://1234.com");	2, 4, 6, 8

With equivalence partitioning, 0 conditions are not met.
Multiple-condition coverage result for equivalence partitioning: 8/8 = 100%

Multiple-condition coverage in boundary analysis		
EC	Test Case	Meet with objective
2a	pb.retrieveDetails(s, "httpp://1234.com");	2, 3

The boundary value analysis of retrieveDetails() is quite similar to updateDetails(). Only EC2 have the different test cases with its boundary values, all the other ECs' test case are the same as equivalence partitioning test. However, since the most of the test cases for EC2 is identical to the EC2 of updateDetails(), we need to avoid to repeat these identical test cases. So only one new test case exists for boundary value analysis. So, the Multiple-condition coverage result for boundary partitioning will be same as the coverage result for equivalence partitioning.
Multiple-condition coverage result for boundary partitioning: 8/8 = 100%

Task 7 Comparison

Firstly, the coverage scores of all methods in the PassBook class for two sets of test cases are the same. This is because both equivalence partitioning and boundary value analysis are derived from equivalence classes, and the derived equivalent classes are non-overlapping, so that enhance the coverage of equivalent partition test cases. The method "loginUser", "updateDetails" and "retriveDetails" get 100% coverage scores (excluding data type error) for two sets of test cases, since the equivalence classes cover all the statements and there are not many boundary values exist for the equivalence classes of these three methods. For the method "addUser", the coverage scores of equivalence partitioning and boundary value analysis is 88.46%, as there are 3 unmet conditions that are logically impossible. Although the coverage scores for both of them are the same, the boundary value analysis covers the additional on-points and off-points for the value and program count length inputs whereas equivalence partitioning did not specifically cover any boundary points. Moreover, for killing mutant, the BoundaryTests are able to kill all five mutants that created in Task 6, the PartitioningTests can only kill 'mutant 2' and 'mutant 4'. Equivalence class and boundary value analysis are complementary as the boundary of input domain will be hard to find without defining the equivalence class first, hence both of them are important. However, boundary value analysis is more efficient than equivalence partitioning in defining more specific boundary and then detecting the transition of the boundary. **Therefore, the boundary value analysis is more effective than equivalence partitioning.**