

# Testing Analysis

Working Period: 12-13/01/2026

Author: Ester Shumeli

Accepted by: Ari Gjerazi, Jurgen Cama

## 1. BVT (Boundary Value Testing)

Method Under Test: isValidPassword(String password, Response objResponse)

```
private static void isValidPassword(String password, Response objResponse) {  
    if (password == null || password.length() < 3) {  
        objResponse.messagesList.add(  
            new Message("Password is not valid, provide valid password with at least  
3 characters.", MessageType.Error));  
    }  
}
```

Package and class:

```
✓ model.validators  
  > CommonValidator.java  
.. --
```

Purpose: Validates a password and appends an error message to objResponse.messagesList if invalid.

1) Test Design Technique: Boundary Value Testing (BVT)

password is considered **valid** when: password != null AND password.length() >= 3

password is **invalid** when: password == null OR password.length() < 3

Boundary : The critical boundary is at **length = 3**.

Boundary	Password Example	Length	Expected Behavior
Min-1	"ab"	2	Add error message
Min	"abc"	3	No error message
Min+1	"abcd"	4	No error message
Nominal valid	"Password1"	9	No error message
Special case	null	—	Add error message

Preconditions (for all cases):

objResponse is not null

objResponse.messagesList is initialized and empty before calling the method

Test Case ID	Input password	Expected Result
TC-PASS-001	null	messagesList size increases by 1; contains error message
TC-PASS-002	"ab"	messagesList size increases by 1; contains error message
TC-PASS-003	"abc"	messagesList unchanged; no new message added
TC-PASS-004	"abcd"	messagesList unchanged; no new message added
TC-PASS-005	"Password1"	messagesList unchanged; no new message added

## dationTest.java - Eclipse IDE

indow Help

The screenshot shows the Eclipse IDE interface with the title bar "dationTest.java - Eclipse IDE". The menu bar includes "Window" and "Help". Below the menu is a toolbar with various icons. The main workspace displays a Java code editor containing a test class. The code is color-coded for syntax, with packages, imports, and annotations in different colors. The class `PasswordValidationTest` contains a setup method and a test method `testPasswordIsNull`. The test method creates a `UserDTO` object with a null password, calls a validation method, and then asserts that there is one message in the response's message list, which is an error message about a password being invalid.

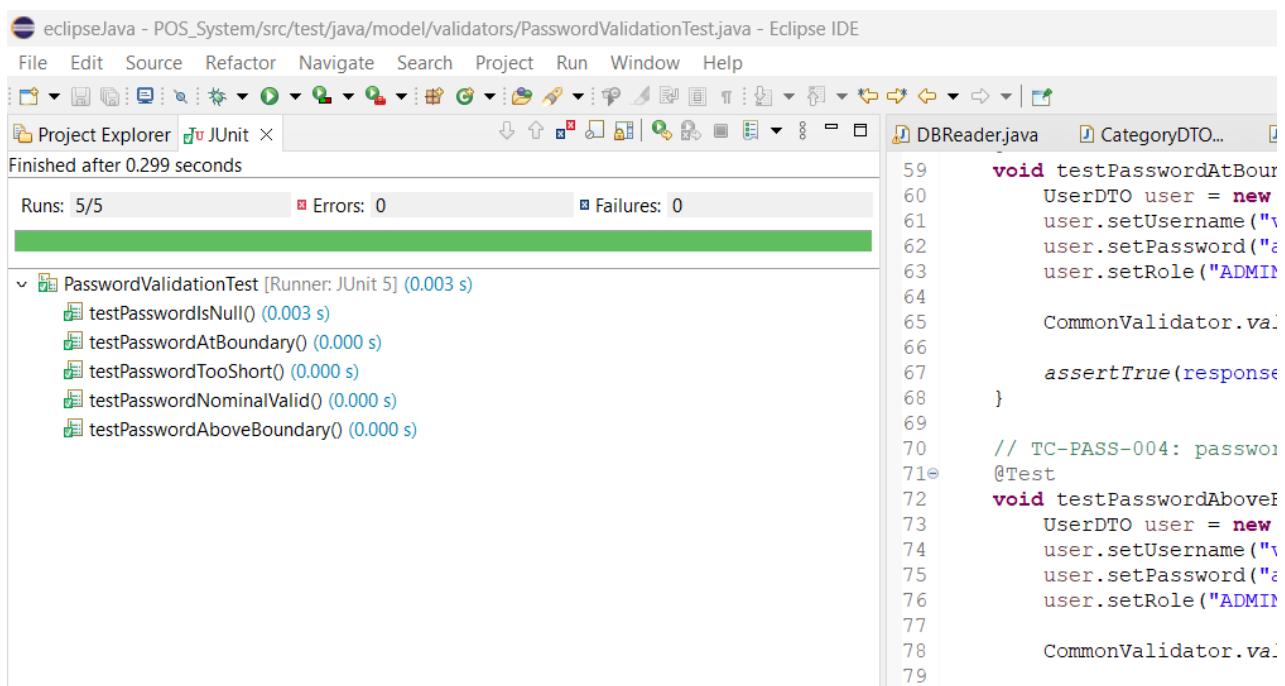
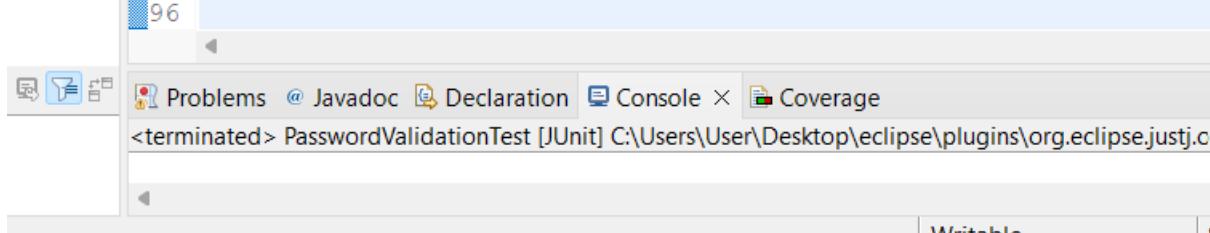
```
1 package model.validators;
2
3import static org.junit.jupiter.api.Assertions.*;
4
5import org.junit.jupiter.api.BeforeEach;
6import org.junit.jupiter.api.Test;
7
8import model.dto.Message;
9import model.dto.MessageType;
10import model.dto.Response;
11import model.dto.UserDTO;
12
13public class PasswordValidationTest {
14
15    private Response response;
16
17    @BeforeEach
18    void setUp() {
19        response = new Response();
20        response.messagesList.clear();
21    }
22
23    // TC-PASS-001: password = null
24    @Test
25    void testPasswordIsNull() {
26        UserDTO user = new UserDTO();
27        user.setUsername("validUser");
28        user.setPassword(null);
29        user.setRole("ADMIN");
30
31        CommonValidator.validateObject(user, response);
32
33        assertEquals(1, response.messagesList.size());
34
35        Message msg = response.messagesList.get(0);
36        assertEquals(MessageType.Error, msg.type());
37        assertEquals(
38            "Password is not valid, provide valid password with at least 3 characters.",
39            msg.message
40        );
41    }
42}
```

```
42
43 // TC-PASS-002: password length < 3
44 @Test
45 void testPasswordTooShort() {
46     UserDTO user = new UserDTO();
47     user.setUsername("validUser");
48     user.setPassword("ab");
49     user.setRole("ADMIN");
50
51     CommonValidator.validateObject(user, response);
52
53     assertEquals(1, response.messagesList.size());
54     assertEquals(MessageType.Error, response.messagesList.get(0).type);
55 }
56
57 // TC-PASS-003: password length == 3 (boundary)
58 @Test
59 void testPasswordAtBoundary() {
60     UserDTO user = new UserDTO();
61     user.setUsername("validUser");
62     user.setPassword("abc");
63     user.setRole("ADMIN");
64
65     CommonValidator.validateObject(user, response);
66
67     assertTrue(response.messagesList.isEmpty());
68 }
```

```

69
70     // TC-PASS-004: password length > 3
71@Test
72 void testPasswordAboveBoundary() {
73     UserDTO user = new UserDTO();
74     user.setUsername("validUser");
75     user.setPassword("abcd");
76     user.setRole("ADMIN");
77
78     CommonValidator.validateObject(user, response);
79
80     assertTrue(response.messagesList.isEmpty());
81 }
82
83 // TC-PASS-005: nominal valid password
84@Test
85 void testPasswordNominalValid() {
86     UserDTO user = new UserDTO();
87     user.setUsername("validUser");
88     user.setPassword("Password123");
89     user.setRole("ADMIN");
90
91     CommonValidator.validateObject(user, response);
92
93     assertTrue(response.messagesList.isEmpty());
94 }
95
96

```



## 2. Equivalence Class Testing (ECT)

### Method Under Test:

```

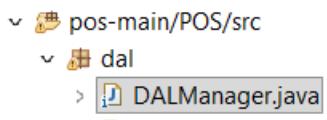
public ArrayList<CustomerDTO> searchCustomersByName(String searchName, Response res)
{
    Connection connection = mySQL.getConnection();
    if (connection == null) {
        Message message = new Message("Database Connection issue please contact
customer services.", MessageType.Exception);
        res.messagesList.add(message);
        return null;
    }

    String query = "SELECT * FROM customers WHERE name LIKE '%" + searchName + "%'";
    ResultSet resultSet = objReader.getRecords(connection, res, query);
    return objMapper.getCustomers(resultSet);

}

```

### Package and class:



EC	Input type	Example	Expected result
EC1	null	null	should be handled
EC2	Empty string	""	returns <b>all customers</b> (because LIKE '%%%') → non-empty list (if DB has customers)
EC3	Valid name that exists	"Ana"	<b>non-empty</b> list
EC4	Valid name that doesn't exist	"zzzz"	<b>empty</b> list
EC5	Name with spaces	"John Doe"	works, returns match/empty
EC6	Special characters / SQL metacharacters	"%' OR '1='1" or "O'Reilly"	should <b>not crash</b> ; ideally should not inject
EC7	Very long string	300+ chars	should not crash; likely empty list or handled

EC	Condition	Expected
EC-DB1	connection == null	return null + add MessageType.Exception

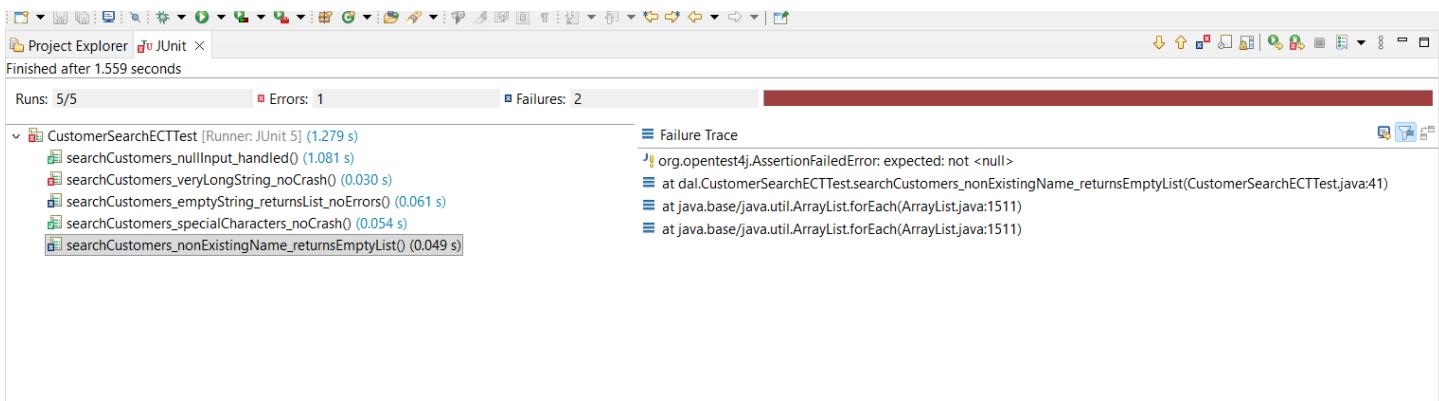
The screenshot shows an IDE interface with a toolbar at the top and a code editor window below. The code editor displays Java test code for a class named `CustomerSearchECTTest`. The code includes imports for `DALManager`, `ArrayList`, `Assertions`, `CustomerDTO`, `Message`, `MessageType`, and `Response`. It contains two test methods: `searchCustomers_emptyString_returnsList_noErrors()` and `searchCustomers_nonExistingName_returnsEmptyList()`. The code uses annotations like `@BeforeEach` and `@Test`. The code editor has line numbers on the left and syntax highlighting for Java keywords and comments.

```
1 package dal;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import java.util.ArrayList;
6
7 import org.junit.jupiter.api.BeforeEach;
8 import org.junit.jupiter.api.Test;
9
10 import model.dto.CustomerDTO;
11 import model.dto.Message;
12 import model.dto.MessageType;
13 import model.dto.Response;
14
15 public class CustomerSearchECTTest {
16
17     private DALManager dal;
18     private Response res;
19
20     @BeforeEach
21     void setUp() {
22         dal = new DALManager();
23         res = new Response();
24     }
25
26     // EC2: empty string
27     @Test
28     void searchCustomers_emptyString_returnsList_noErrors() {
29         ArrayList<CustomerDTO> result = dal.searchCustomersByName("", res);
30
31         // If DB connection works, result should be a list (possibly empty)
32         assertNotNull(result, "Expected a non-null list when DB connection is available");
33         assertTrue(res.isSuccessful(), "No Error/Exception messages expected");
34     }
35
36     // EC4: valid non-existing name -> empty list
37     @Test
38     void searchCustomers_nonExistingName_returnsEmptyList() {
```

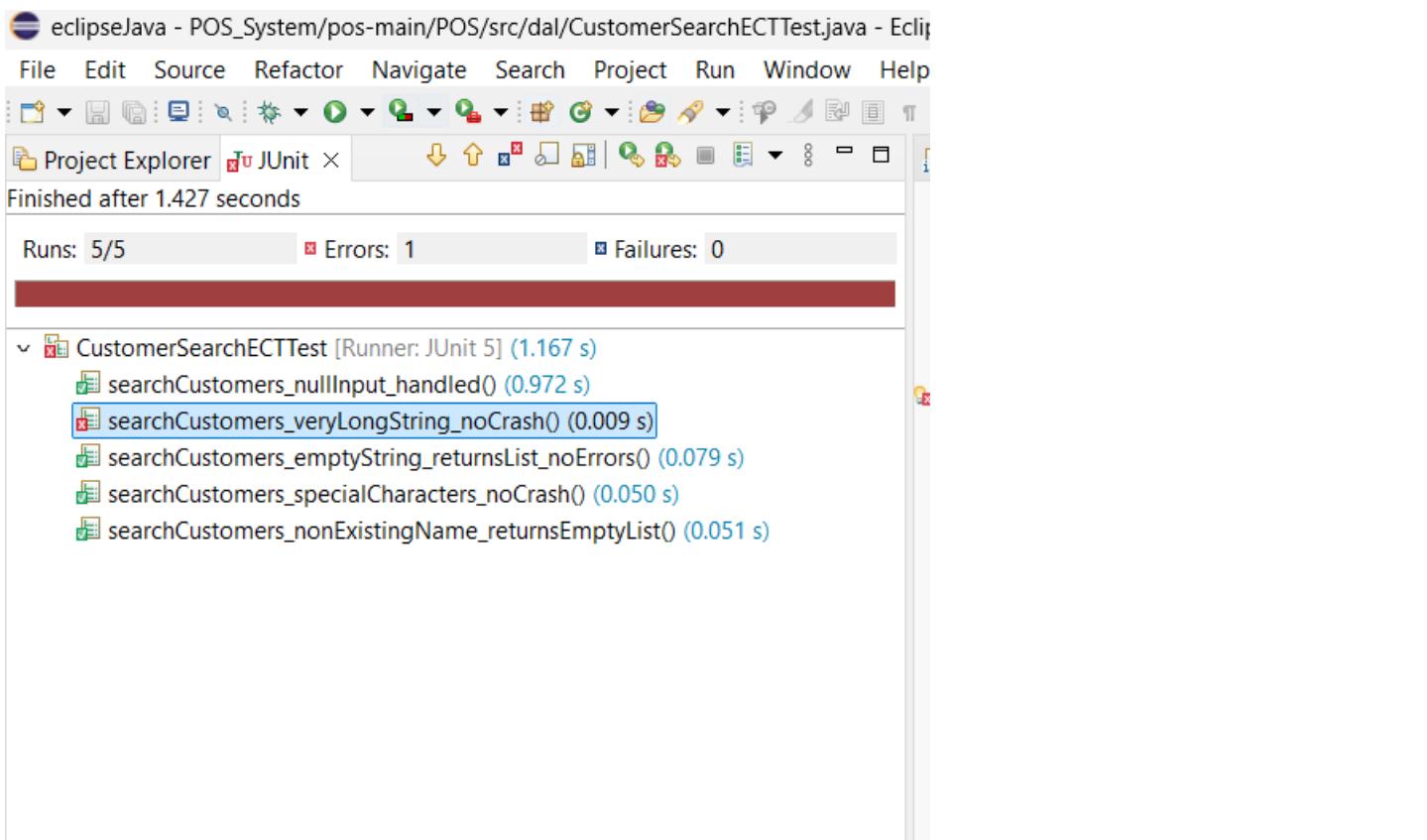
Problems @ Javadoc Declaration Console × Coverage

```
33
36 // EC4: valid non-existing name -> empty list
37 @Test
38 void searchCustomers_nonExistingName_returnsEmptyList() {
39     ArrayList<CustomerDTO> result = dal.searchCustomersByName("zzzz_non_existing_12345", res);
40
41     assertNotNull(result);
42     assertTrue(result.isEmpty(), "Expected no matches for a non-existing name");
43     assertTrue(res.isSuccessful());
44 }
45
46 // EC5: special characters -> robustness (should not crash)
47 @Test
48 void searchCustomers_specialCharacters_noCrash() {
49     ArrayList<CustomerDTO> result = dal.searchCustomersByName("@#$%^&*", res);
50
51     // Robustness expectation: no crash.
52     // Acceptable outcomes:
53     // 1) returns list (possibly empty) and success
54     // 2) returns null and records an Exception/Error message
55     if (result == null) {
56         assertFalse(res.messagesList.isEmpty(), "If result is null, an error/exception message should exist");
57         assertFalse(res.isSuccessful(), "Should be unsuccessful if an exception/error occurred");
58     } else {
59         assertNotNull(result);
60         // may be empty or non-empty depending on DB content
61     }
62 }
63
64 // EC6: very long string -> robustness
65 @Test
66 void searchCustomers_veryLongString_noCrash() {
67     String longName = "a".repeat(300);
68
69     ArrayList<CustomerDTO> result = dal.searchCustomersByName(longName, res);
70
71     // Same robustness expectation
72 }
```

```
66 void searchCustomers_veryLongString_noCrash() {
67     String longName = "a".repeat(300);
68
69     ArrayList<CustomerDTO> result = dal.searchCustomersByName(longName, res);
70
71     // Same robustness expectation
72     if (result == null) {
73         assertFalse(res.messagesList.isEmpty());
74         assertFalse(res.isSuccessful());
75     } else {
76         assertNotNull(result);
77     }
78 }
79
80 // EC1: null input
81 @Test
82 void searchCustomers_nullInput_handled() {
83     ArrayList<CustomerDTO> result = dal.searchCustomersByName(null, res);
84
85     // SQL string concatenation turns null into "null" inside the query,
86     // so often it won't crash and returns list (maybe empty).
87     // If it DOES fail (e.g., DBReader rejects it), then it should record error/exception.
88     if (result == null) {
89         assertFalse(res.messagesList.isEmpty());
90         assertFalse(res.isSuccessful());
91     } else {
92         assertNotNull(result);
93     }
94 }
95
96
97 }
98 }
```



The results are as expected because I haven't run MySQL and the database POS is not connected, but the logic of the testing is precise and clear.



customers 1					
Action Output			Output		
#	Time	Action	Message	Duration / Fetch	
✓ 1	21:37:09	CREATE DATABASE IF NOT EXISTS pos	1 row(s) affected	0.016 sec	
✓ 2	21:37:09	USE pos	0 row(s) affected	0.016 sec	
✓ 3	21:37:44	USE pos	0 row(s) affected	0.000 sec	
✓ 4	21:37:44	CREATE TABLE IF NOT EXISTS customers ( id INT AUTO_INCREMENT...	0 row(s) affected	0.063 sec	
✓ 5	21:38:01	INSERT INTO customers (name, phoneNumber) VALUES ('Ana Shumeli', '06...	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.016 sec	
✓ 6	21:38:14	SELECT * FROM customers LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec	

I connected it to the DB Pos using MySqlLite Workbench 8.0 and the failures do not appear anymore.

The only error is the repeat() method ->**Java version issue**, not a testing mistake

String.repeat(int) **exists only from Java 11 onward**.

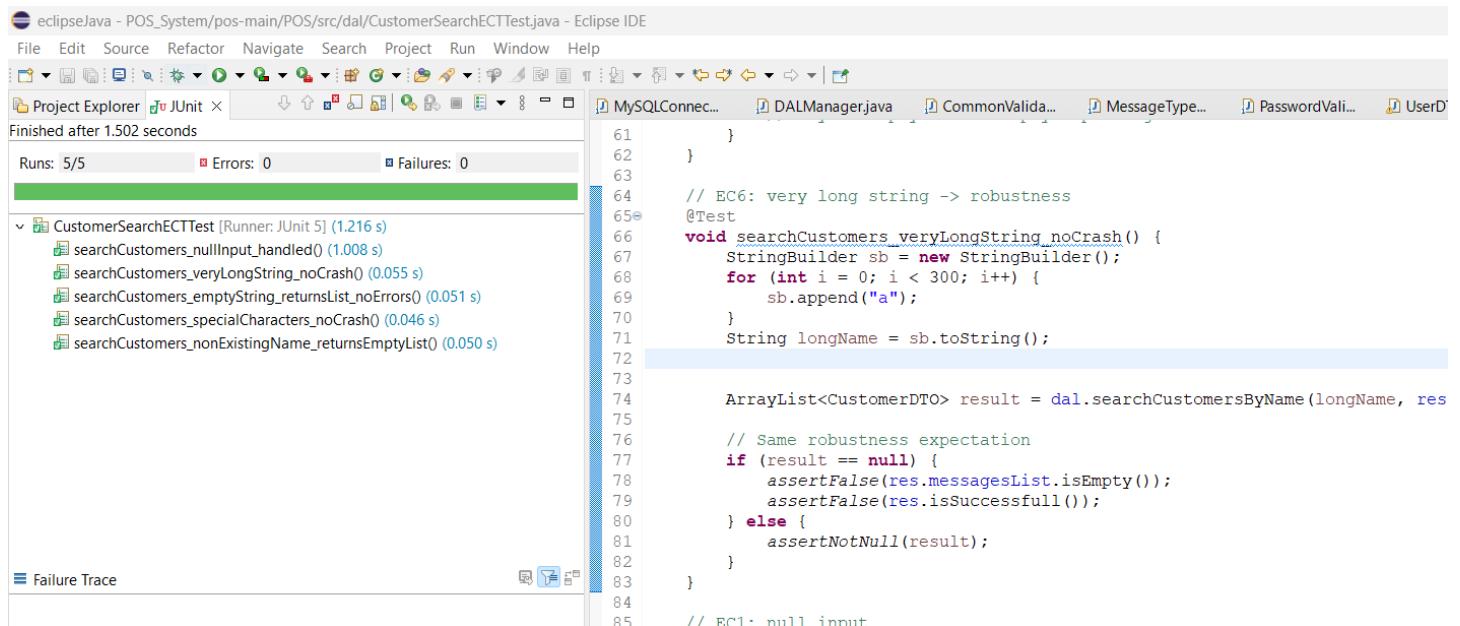
Our project is currently using **Java 8**, which is why Eclipse marks repeat in red.

I'll try to fix it using append and StringBuilder:

```
// EC6: very long string -> robustness
@Test
void searchCustomers_veryLongString_noCrash() {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < 300; i++) {
        sb.append("a");
    }
    String longName = sb.toString();

    ArrayList<CustomerDTO> result = dal.searchCustomersByName(
        longName);

    // Same robustness expectation
    if (result == null) {
        assertFalse(res.messagesList.isEmpty());
    }
}
```



### 3. Code Coverage + MC/DC

#### Method Under Test:

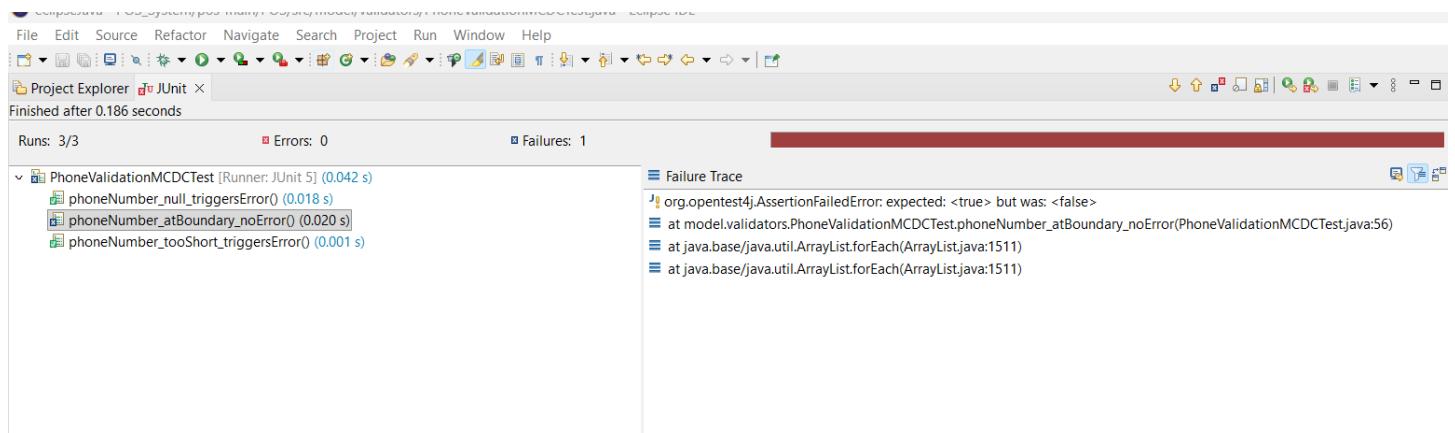
```
private static void isValidPhoneNo(String phoneNumber, Response response) {
    if (phoneNumber == null || phoneNumber.length() < 10) {
        response.messagesList.add(
            new Message("Phone Number is not valid, provide valid Phone Number with
at least 10 characters.", MessageType.Error));
    }
}
```

#### Package and class:

model.validators  
CommonValidator.java

Test	phoneNumber	A: null?	B: len<10?	Decision (add error?)	Proves
T1	null	T	-	TRUE	A alone makes decision true
T2	"123456789" (9 chars)	F	T	TRUE	B alone makes decision true
T3	"1234567890" (10 chars)	F	F	FALSE	Both false → decision false

1<sup>st</sup> try:



The screenshot shows a Java code editor with a toolbar at the top and several tabs open in the background. The main code area displays a test class named `PhoneValidationMCDCTest`. The code uses JUnit annotations like `@BeforeEach` and `@Test` to define two test methods: `phoneNumber_null_triggersError()` and `phoneNumber_tooShort_triggersError()`. Both tests create a `CustomerDTO` object with a name set to "Ana" and a null or short phone number respectively, then call `CommonValidator.validateObject(c, res)`. The test asserts that the response is not successful and contains an error message type.

```
1 package model.validators;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.BeforeEach;
6 import org.junit.jupiter.api.Test;
7
8 import model.dto.CustomerDTO;
9 import model.dto.MessageType;
10 import model.dto.Response;
11
12 public class PhoneValidationMCDCTest {
13
14     private Response res;
15
16     @BeforeEach
17     void setUp() {
18         res = new Response();
19     }
20
21     // T1 (MC/DC for A): phoneNumber == null => error
22     @Test
23     void phoneNumber_null_triggersError() {
24         CustomerDTO c = new CustomerDTO();
25         c.setName("Ana");
26         c.setPhoneNumber(null);
27
28         CommonValidator.validateObject(c, res);
29
30         assertFalse(res.isSuccessfull());
31         assertEquals(MessageType.Error, res.messagesList.get(0).type);
32     }
33
34     // T2 (MC/DC for B): non-null but length < 10 => error
35     @Test
36     void phoneNumber_tooShort_triggersError() {
37         CustomerDTO c = new CustomerDTO();
38         c.setName("Ana");
39         c.setPhoneNumber("123456789"); // 9 chars
```

```

3
4 // T2 (MC/DC for B): non-null but length < 10 => error
5 @Test
6 void phoneNumber_tooShort_triggersError() {
7     CustomerDTO c = new CustomerDTO();
8     c.setName("Ana");
9     c.setPhoneNumber("123456789"); // 9 chars
0
1     CommonValidator.validateObject(c, res);
2
3     assertFalse(res.isSuccessfull());
4     assertEquals(MessageType.Error, res.messagesList.get(0).type);
5 }
6
7 // T3 (Both false): length == 10 => no error
8 @Test
9 void phoneNumber_atBoundary_noError() {
0     CustomerDTO c = new CustomerDTO();
1     c.setName("Ana");
2     c.setPhoneNumber("1234567890"); // 10 chars
3
4     CommonValidator.validateObject(c, res);
5
6     assertTrue(res.isSuccessfull());
7 }
8
9

```

Problems @ Javadoc Declaration Console × Coverage

1 failure-> When testing one decision, all other inputs must be valid.

So: Let's give the customer a clearly valid name.

During MC/DC testing of the phone number validation condition, all other input fields were fixed to valid values to ensure that only the phone number conditions independently affected the decision outcome.

So, I changed line 51

```

47 // T3 (Both false): length == 10 => no error
48 @Test
49 void phoneNumber_atBoundary_noError() {
50     CustomerDTO c = new CustomerDTO();
51     c.setName("ValidCustomerName");
52     c.setPhoneNumber("1234567890"); // 10 chars
53
54     CommonValidator.validateObject(c, res);
55
56     assertTrue(res.isSuccessfull());
57 }
58
59

```

Problems @ Javadoc Declaration Console × Coverage

<terminated> PhoneValidationMCDCTest [JUnit] C:\Users\User\Desktop\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x

