

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



```
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.Message;
9 import model.dto.MessageType;
10 import model.dto.Response;
11 import model.dto.UserDTO;
12
13 public 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.",
```

```

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

```

Problems @ Javadoc Declaration Console × Coverage
<terminated> PasswordValidationTest [JUnit] C:\Users\User\Desktop\eclipse\plugins\org.eclipse.justj.c

eclipseJava - POS_System/src/test/java/model/validators/PasswordValidationTest.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer JUnit ×

Finished after 0.299 seconds

Runs: 5/5

Errors: 0

Failures: 0

PasswordValidationTest [Runner: JUnit 5] (0.003 s)

testPasswordIsNull() (0.003 s)

testPasswordAtBoundary() (0.000 s)

testPasswordTooShort() (0.000 s)

testPasswordNominalValid() (0.000 s)

testPasswordAboveBoundary() (0.000 s)

```

59 void testPasswordAtBour
60 UserDTO user = new
61 user.setUsername("\
62 user.setPassword("\
63 user.setRole("ADMIN
64
65 CommonValidator.val
66
67 assertTrue(response
68 }
69
70 // TC-PASS-004: passwo
71 @Test
72 void testPasswordAbove
73 UserDTO user = new
74 user.setUsername("\
75 user.setPassword("\
76 user.setRole("ADMIN
77
78 CommonValidator.val
79

```

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;
    }

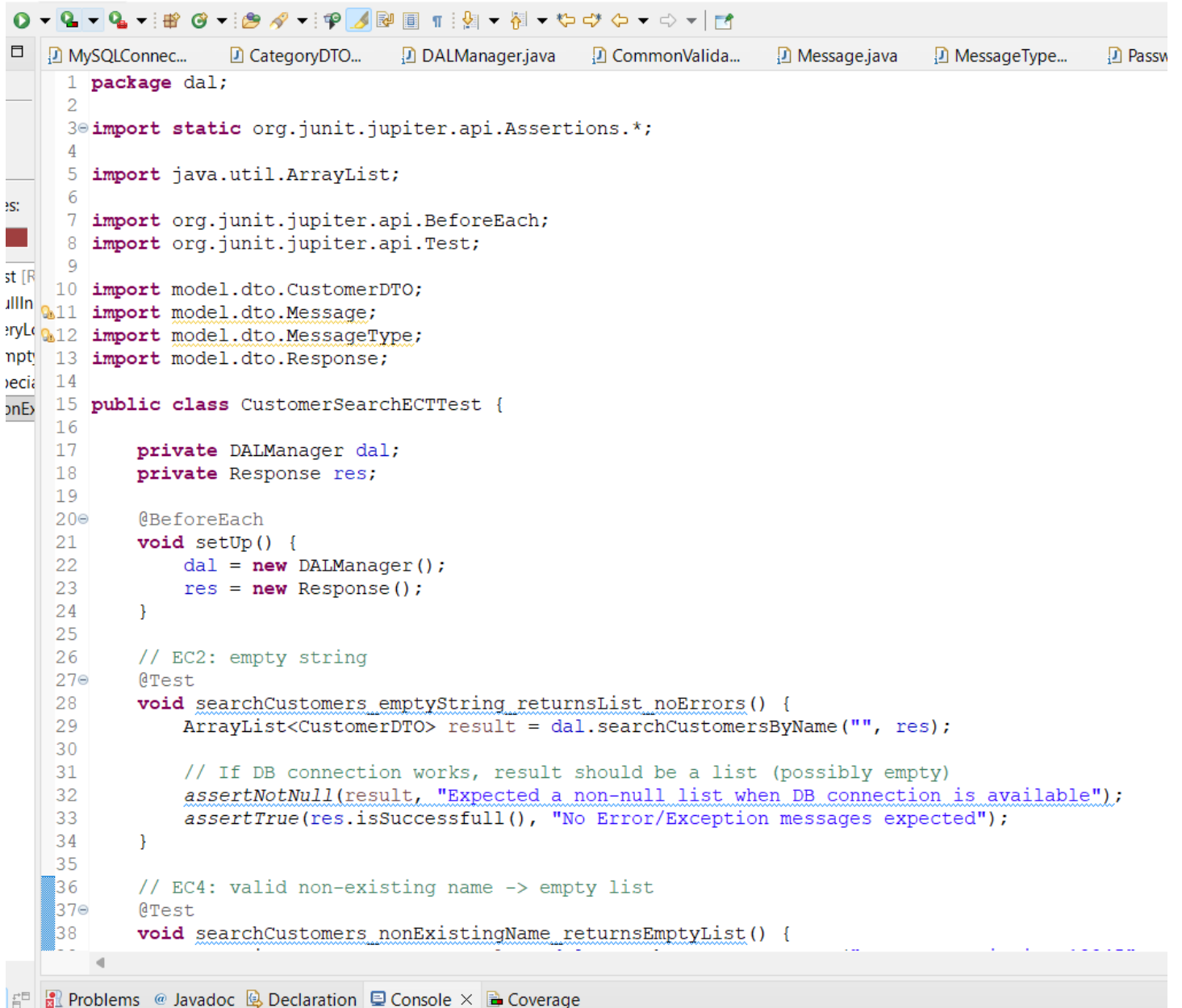
    String query = "SELECT * FROM customers WHERE name LIKE '%" + searchName + "%'";
    ResultSet resultSet = objReader.getRecords(connection, res, query);
    return objMapper.getCustomers(resultSet);
}
```

Package and class:

```
pos-main/POS/src
dal
DALManager.java
```

EC	Input type	Example	Expected result
EC1	null	null	should be handled
EC2	Empty string	""	returns all customers (because LIKE '%%') → non-empty list (if DB has customers)
EC3	Valid name that exists	"Ana"	non-empty list
EC4	Valid name that doesn't exist	"zzzz"	empty list
EC5	Name with spaces	"John Doe"	works, returns match/empty
EC6	Special characters / SQL metacharacters	"%' OR '1'='1" or "O'Reilly"	should not crash ; ideally should not inject (but your code is vulnerable)
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 window with a Java test class named `CustomerSearchECTest`. The code includes imports for JUnit, ArrayList, and DTOs. It features a `setUp` method and two test methods: `searchCustomers_emptyString_returnsList_noErrors` and `searchCustomers_nonExistingName_returnsEmptyList`. The IDE interface includes a toolbar at the top, a tab bar with files like `MySQLConnec...`, `CategoryDTO...`, `DALManager.java`, `CommonValida...`, `Message.java`, `MessageType...`, and `Passw...`, and a bottom panel with tabs for `Problems`, `Javadoc`, `Declaration`, `Console`, and `Coverage`.

```
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 CustomerSearchECTest {
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.isSuccessfull(), "No Error/Exception messages expected");
34     }
35
36     // EC4: valid non-existing name -> empty list
37     @Test
38     void searchCustomers_nonExistingName_returnsEmptyList() {
```

```

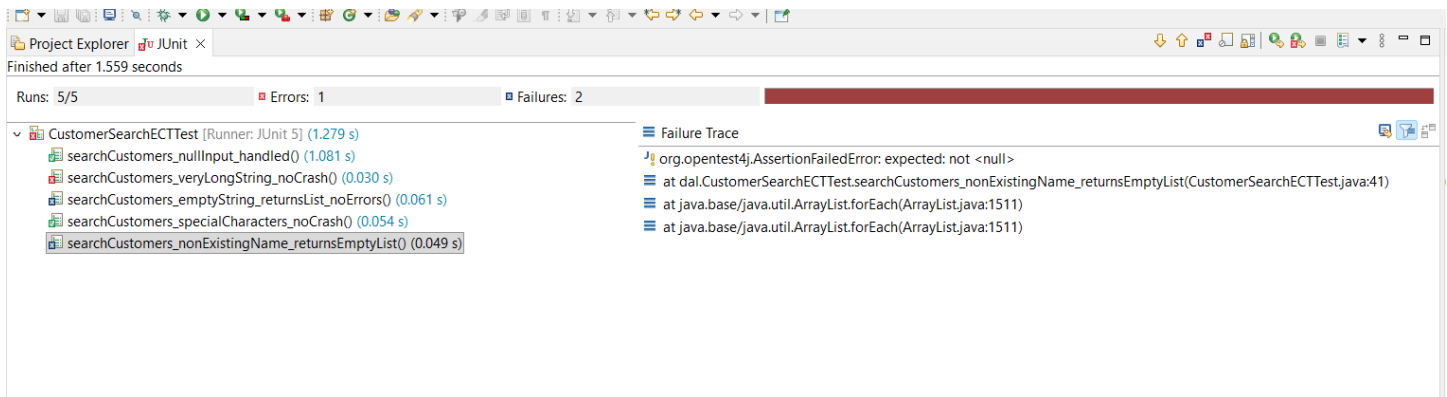
36 // EC4: valid non-existing name -> empty list
37 @Test
38 void searchCustomers_nonExistingName_returnsEmptyList() {
39     ArrayList<CustomerDTO> result = dal.searchCustomersByName("zzzz_non_existing_12345", res);
40
41     assertNotNull(result);
42     assertTrue(result.isEmpty(), "Expected no matches for a non-existing name");
43     assertTrue(res.isSuccessfull());
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.isSuccessfull(), "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

```

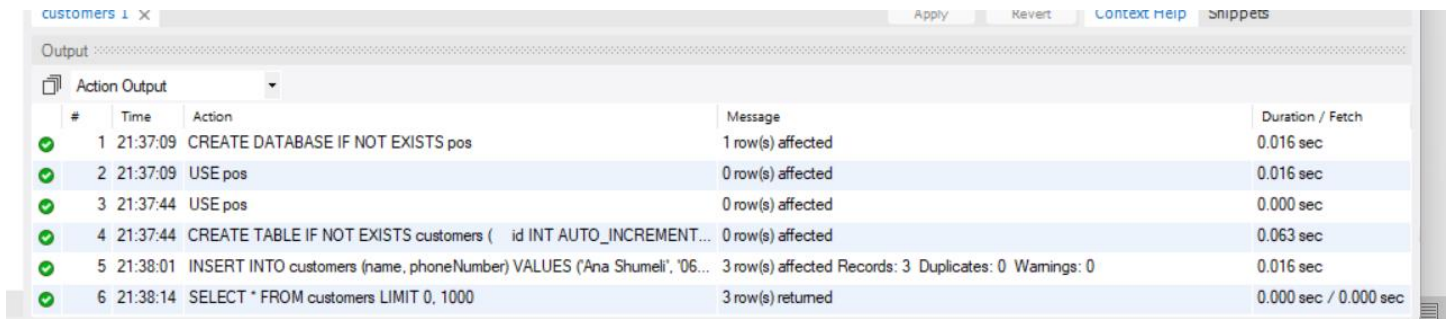
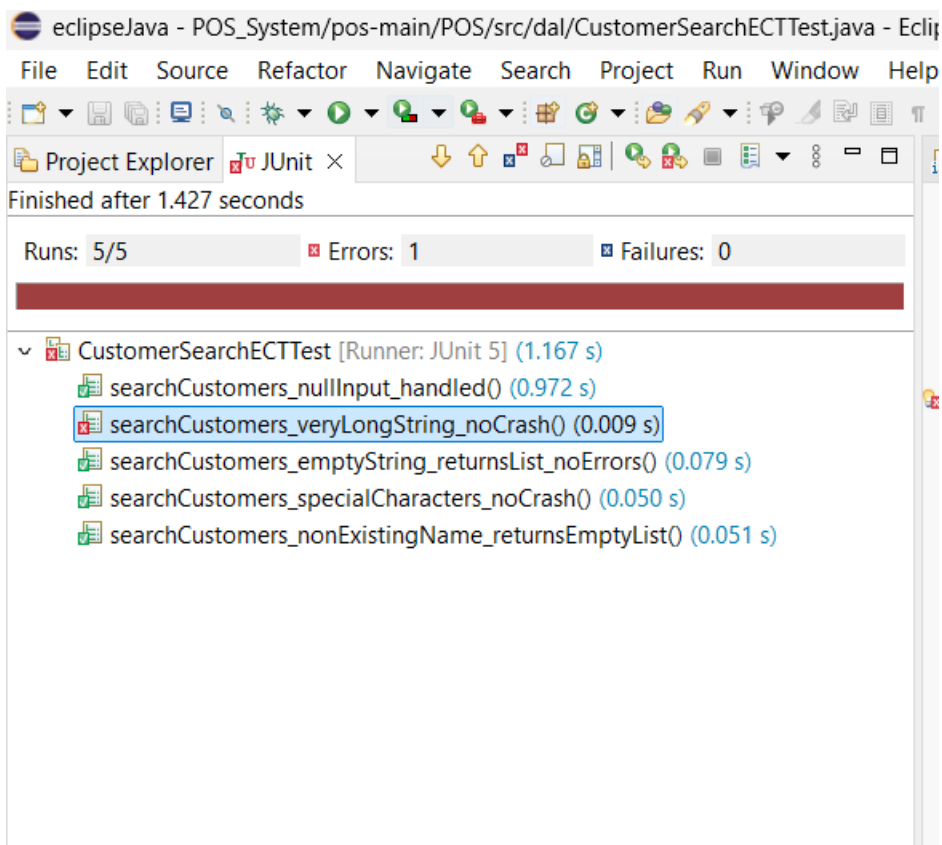
```

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.isSuccessfull());
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.isSuccessfull());
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.



I connected it to the DB Pos using MySQLite 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, res);
```

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

The screenshot shows the Eclipse IDE interface. The top toolbar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. Below the toolbar is the Project Explorer showing the project structure. The main editor displays the source code of CustomerSearchECTTest.java. The code includes a test method searchCustomers_veryLongString_noCrash() that uses a StringBuilder to create a long string of 300 'a's. The test then calls dal.searchCustomersByName() and asserts that the result is not null and that the messagesList is not empty. The bottom of the IDE shows the Run console with the output: Runs: 5/5, Errors: 0, Failures: 0. The test results are listed below the console, showing that all tests passed successfully.

```
eclipse.java - POS_System/pos-main/POS/src/dal/CustomerSearchECTTest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer JUnit X
Finished after 1.502 seconds
Runs: 5/5 Errors: 0 Failures: 0
CustomerSearchECTTest [Runner: JUnit 5] (1.216 s)
  searchCustomers_nullInput_handled() (1.008 s)
  searchCustomers_veryLongString_noCrash() (0.055 s)
  searchCustomers_emptyString_returnsList_noErrors() (0.051 s)
  searchCustomers_specialCharacters_noCrash() (0.046 s)
  searchCustomers_nonExistingName_returnsEmptyList() (0.050 s)
Failure Trace
61 }
62 }
63
64 // EC6: very long string -> robustness
65 @Test
66 void searchCustomers_veryLongString_noCrash() {
67     StringBuilder sb = new StringBuilder();
68     for (int i = 0; i < 300; i++) {
69         sb.append("a");
70     }
71     String longName = sb.toString();
72
73
74     ArrayList<CustomerDTO> result = dal.searchCustomersByName(longName, res);
75
76     // Same robustness expectation
77     if (result == null) {
78         assertFalse(res.messagesList.isEmpty());
79         assertFalse(res.isSuccessfull());
80     } else {
81         assertNotNull(result);
82     }
83 }
84
85 // EC1: null input
```

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

1st try:

IDE Screenshot showing test results for `PhoneValidationMCDCTest`.

Project Explorer: `PhoneValidationMCDCTest` [Runner: JUnit 5] (0.042 s)

- `phoneNumber_null_triggersError()` (0.018 s)
- `phoneNumber_atBoundary_noError()` (0.020 s)
- `phoneNumber_tooShort_triggersError()` (0.001 s)

Run Summary: Runs: 3/3, Errors: 0, Failures: 1

Failure Trace:

```
org.opentest4j.AssertionFailedError: expected: <true> but was: <false>  
    at model.validators.PhoneValidationMCDCTest.phoneNumber_atBoundary_noError(PhoneValidationMCDCTest.java:56)  
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)  
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
```

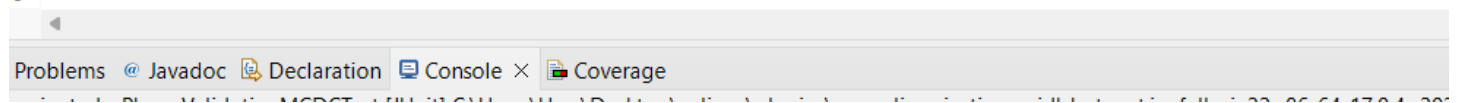
```
Search Project Run Window Help
ObjectMappe... CategoryDTO... ProductDTO.java ObjectRemove... DALManager.java CommonVali
1 package model.validators;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.BeforeEach;
6 import org.junit.jupiter.api.Test;
7
8 import model.dto.CustomerDTO;
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 phoneNumberTooShortTriggersError() {
7     CustomerDTO c = new CustomerDTO();
8     c.setName("Ana");
9     c.setPhoneNumber("123456789"); // 9 chars
10
11     CommonValidator.validateObject(c, res);
12
13     assertFalse(res.isSuccessfull());
14     assertEquals(MessageType.Error, res.messagesList.get(0).type);
15 }
16
17 // T3 (Both false): length == 10 => no error
18 @Test
19 void phoneNumberAtBoundaryNoError() {
20     CustomerDTO c = new CustomerDTO();
21     c.setName("Ana");
22     c.setPhoneNumber("1234567890"); // 10 chars
23
24     CommonValidator.validateObject(c, res);
25
26     assertTrue(res.isSuccessfull());
27 }
28 }
29

```



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

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

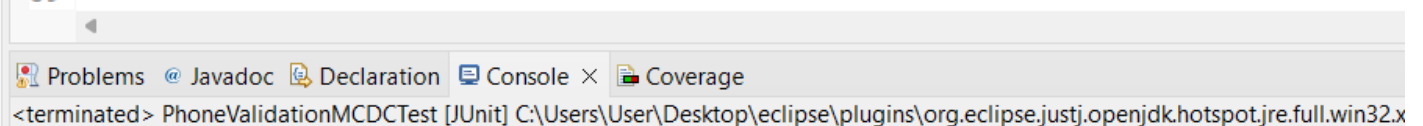
During MC/DC testing of the phone number validation condition, all other input fields were fixed to valid values to ensure that only the phone number conditions independently affected the decision outcome.

So, I changed line 51

```

47 // T3 (Both false): length == 10 => no error
48 @Test
49 void phoneNumberAtBoundaryNoError() {
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

```





Project Explorer JUnit ×

Finished after 0.167 seconds

Runs: 3/3 Errors: 0 Failures: 0

- PhoneValidationMCDCTest [Runner: JUnit 5] (0.001 s) Failure Trace
- phoneNumber_null_triggersError() (0.000 s)
 - phoneNumber_atBoundary_noError() (0.000 s)
 - phoneNumber_tooShort_triggersError() (0.001 s)