Testing Analysis

Working Period: 12-13/01/2026

Author: Ester Shumeli

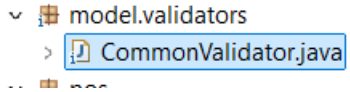
Accepted by: Ari Gjerazi, Jurgen Cama

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:



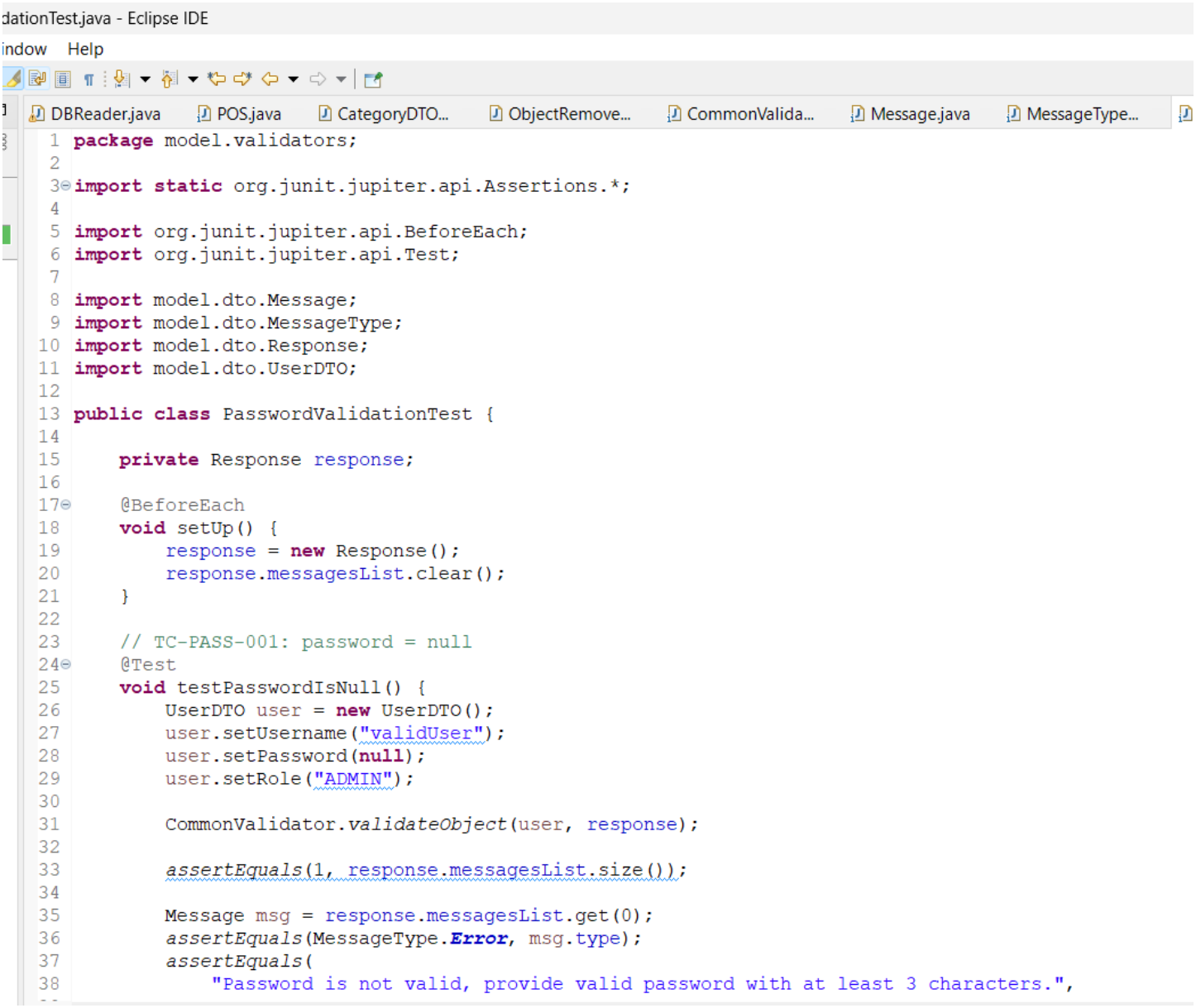
Purpose: Validates a password and appends an error message to objResponse.messagesList if invalid.
1) Test Design Technique: Boundary Value Testing (BVT)

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
DBReader.java
POS.java
CategoryDTO...
ObjectRemove...
CommonValida...
Message.java
MessageType...
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.",

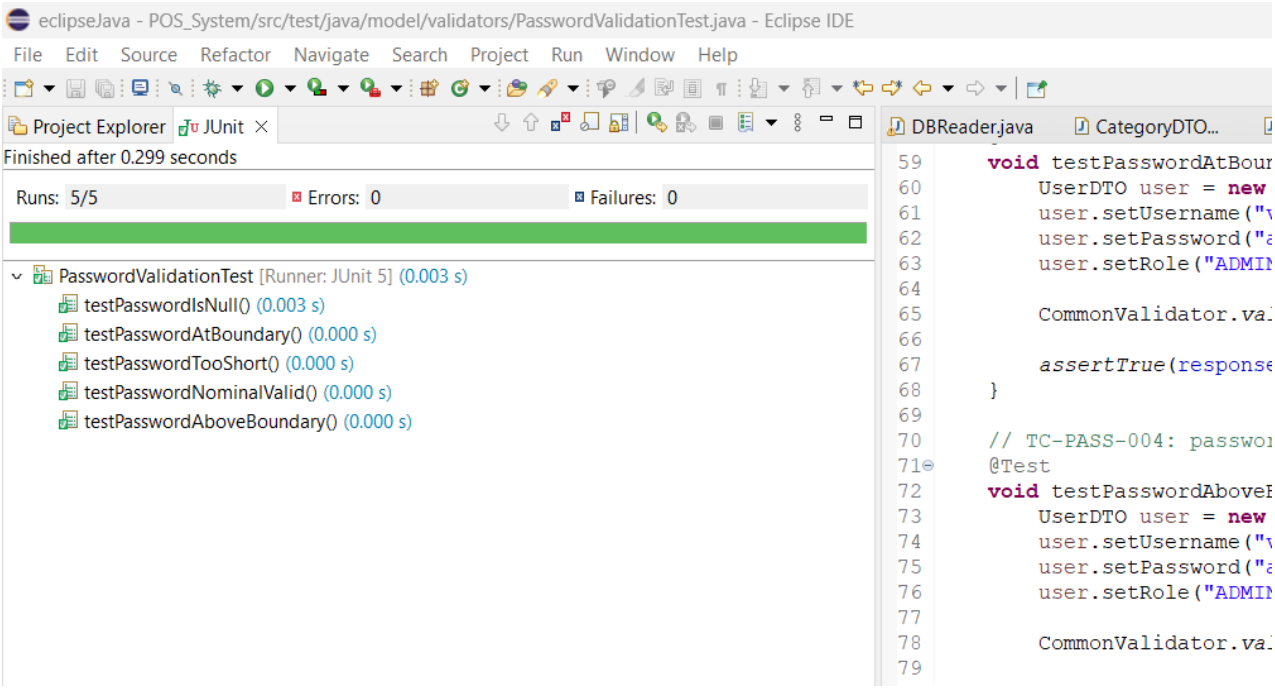
```java
    // TC-PASS-002: password length < 3
    @Test
    void testPasswordTooShort() {
        UserDTO user = new UserDTO();
        user.setUsername("validUser");
        user.setPassword("ab");
        user.setRole("ADMIN");

        CommonValidator.validateObject(user, response);

        assertEquals(1, response.messagesList.size());
        assertEquals(MessageType.Error, response.messagesList.get(0).type);
    }

    // TC-PASS-003: password length == 3 (boundary)
    @Test
    void testPasswordAtBoundary() {
        UserDTO user = new UserDTO();
        user.setUsername("validUser");
        user.setPassword("abc");
        user.setRole("ADMIN");

        CommonValidator.validateObject(user, response);

        assertTrue(response.messagesList.isEmpty());
    }
```

```
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

Writable

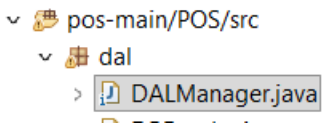


2. Equivalence Class Testing (ECT)

Method Under Test:

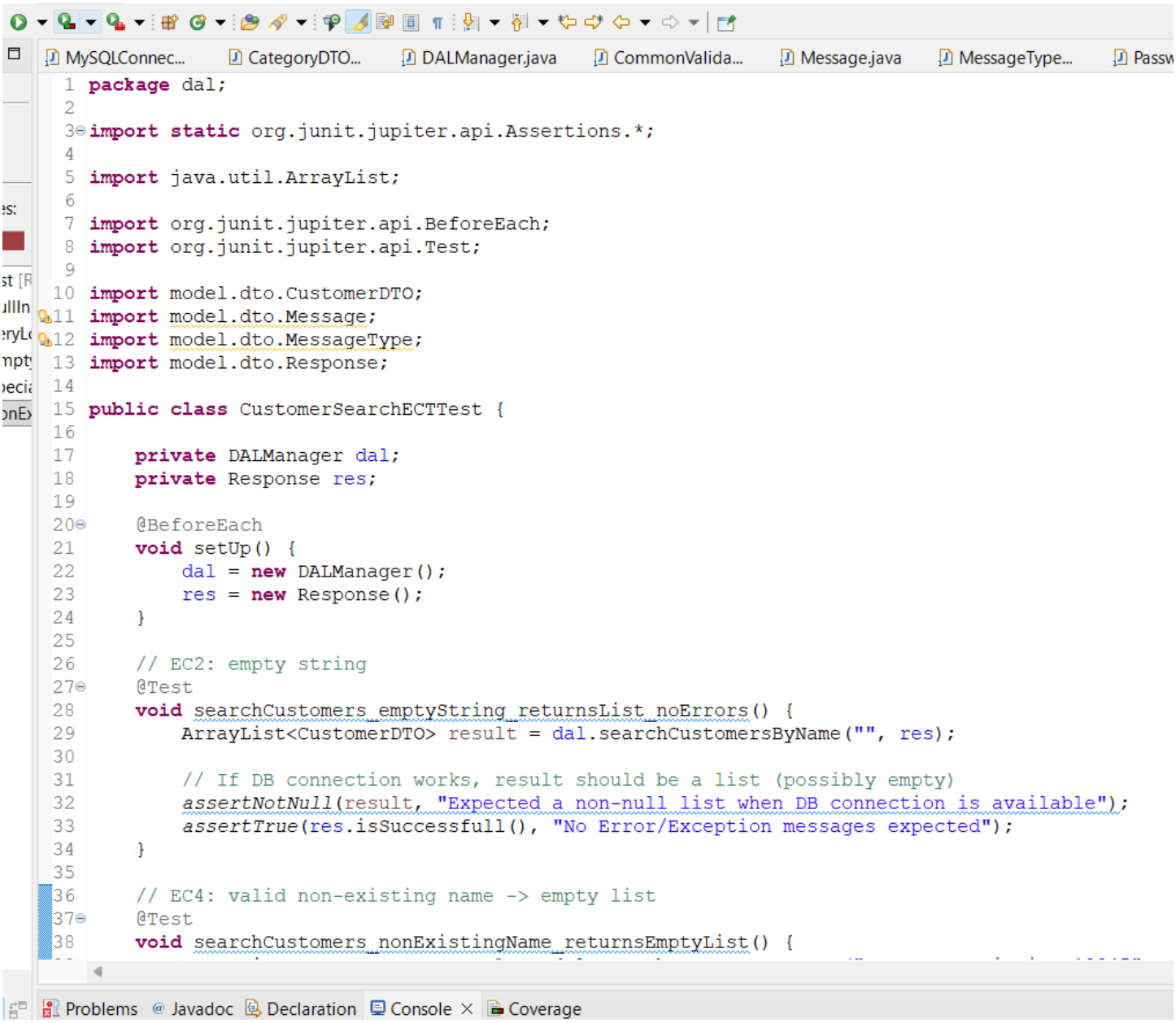
```
    public ArrayList<CustomerDTO> searchCustomersByName(String searchName, Response res)
{
      Connection connection = mySQL.getConnection();
        if (connection == null) {
            Message message = new Message("Database Connection issue please contact
customer services.", MessageType.Exception);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

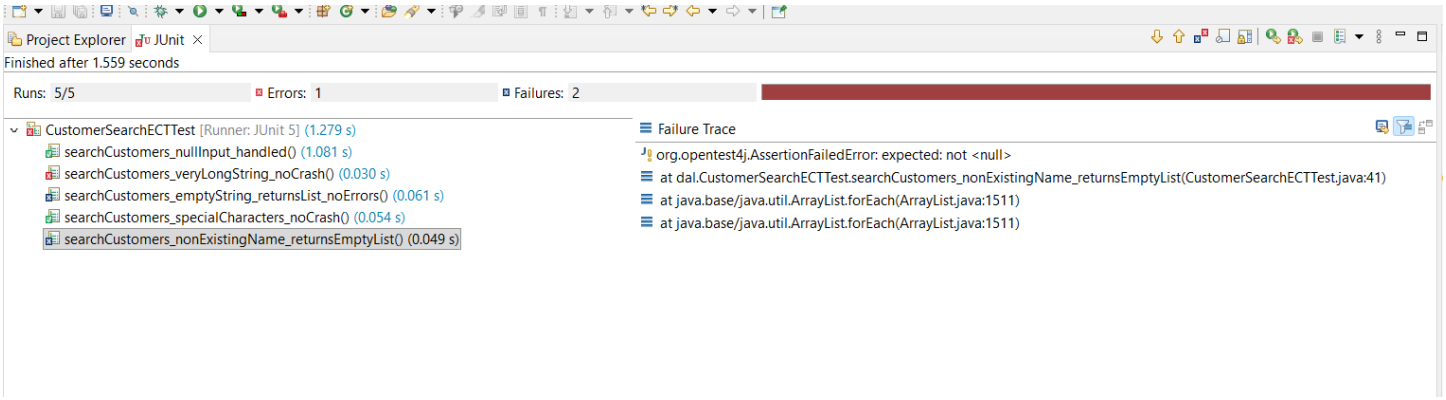
EC	Condition	Expected
EC-DB1	connection == null	return null + add MessageType.Exception



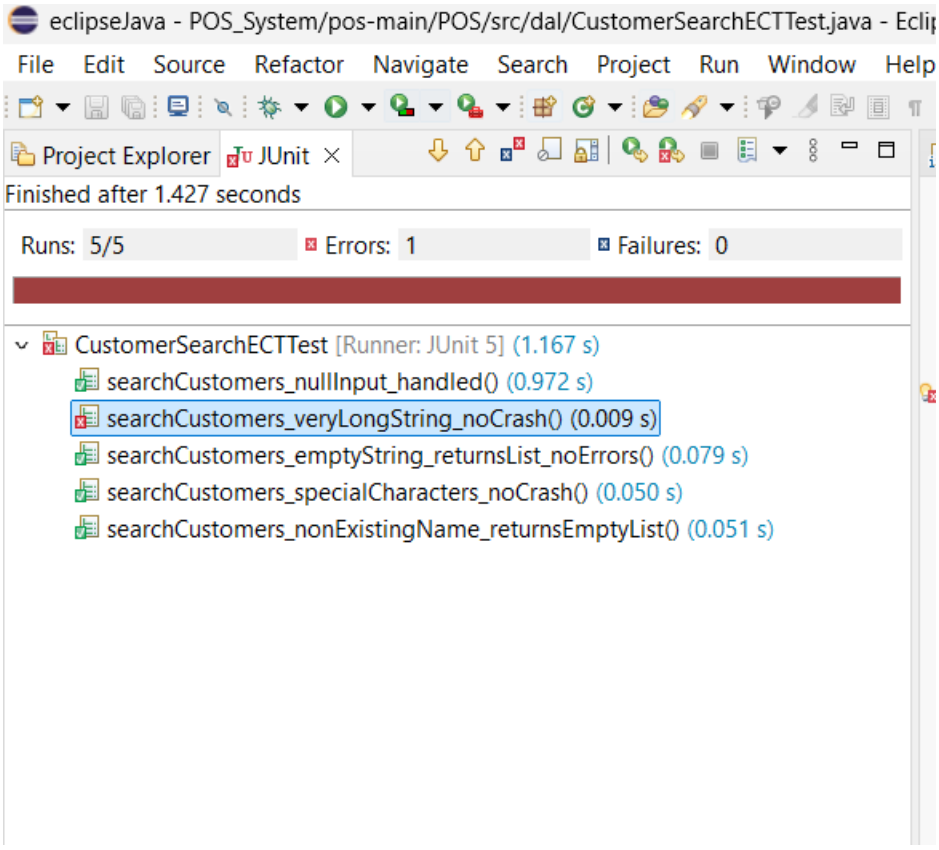

```java
package dal;

import static org.junit.jupiter.api.Assertions.*;

import java.util.ArrayList;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import model.dto.CustomerDTO;
import model.dto.Message;
import model.dto.MessageType;
import model.dto.Response;

public class CustomerSearchECTTest {

    private DALManager dal;
    private Response res;

    @BeforeEach
    void setUp() {
        dal = new DALManager();
        res = new Response();
    }

    // EC2: empty string
    @Test
    void searchCustomers_emptyString_returnsList_noErrors() {
        ArrayList<CustomerDTO> result = dal.searchCustomersByName("", res);

        // If DB connection works, result should be a list (possibly empty)
        assertNotNull(result, "Expected a non-null list when DB connection is available");
        assertTrue(res.isSuccessfull(), "No Error/Exception messages expected");
    }

    // EC4: valid non-existing name -> empty list
    @Test
    void searchCustomers_nonExistingName_returnsEmptyList() {
```

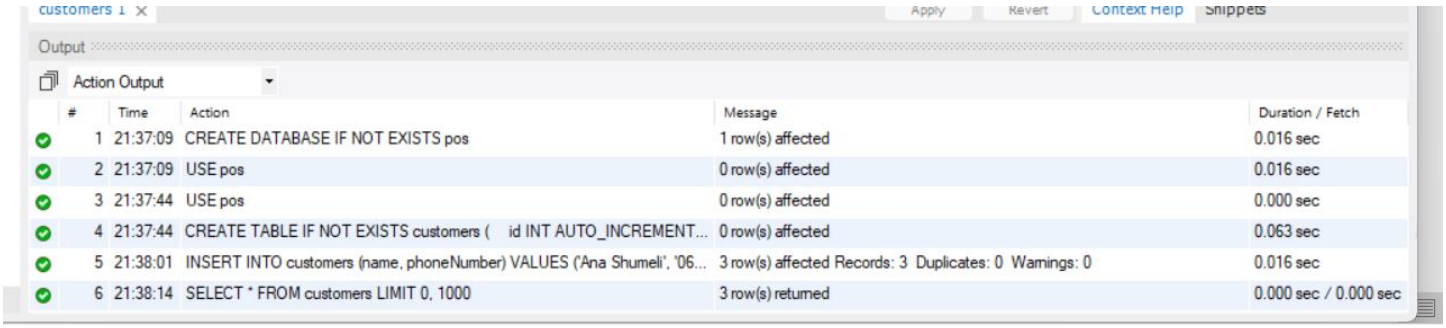
```
    // EC4: valid non-existing name -> empty list
    @Test
    void searchCustomers_nonExistingName_returnsEmptyList() {
        ArrayList<CustomerDTO> result = dal.searchCustomersByName("zzzz_non_existing_12345", res);

        assertNotNull(result);
        assertTrue(result.isEmpty(), "Expected no matches for a non-existing name");
        assertTrue(res.isSuccessfull());
    }

    // EC5: special characters -> robustness (should not crash)
    @Test
    void searchCustomers_specialCharacters_noCrash() {
        ArrayList<CustomerDTO> result = dal.searchCustomersByName("@#$%^&*", res);

        // Robustness expectation: no crash.
        // Acceptable outcomes:
        // 1) returns list (possibly empty) and success
        // 2) returns null and records an Exception/Error message
        if (result == null) {
            assertFalse(res.messagesList.isEmpty(), "If result is null, an error/exception message should exist");
            assertFalse(res.isSuccessfull(), "Should be unsuccessful if an exception/error occurred");
        } else {
            assertNotNull(result);
            // may be empty or non-empty depending on DB content
        }
    }

    // EC6: very long string -> robustness
    @Test
    void searchCustomers_veryLongString_noCrash() {
        String longName = "a".repeat(300);

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

        // Same robustness expectation
```

```
    void searchCustomers_veryLongString_noCrash() {
        String longName = "a".repeat(300);

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

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

    // EC1: null input
    @Test
    void searchCustomers_nullInput_handled() {
        ArrayList<CustomerDTO> result = dal.searchCustomersByName(null, res);

        // SQL string concatenation turns null into "null" inside the query,
        // so often it won't crash and returns list (maybe empty).
        // If it DOES fail (e.g., DBReader rejects it), then it should record error/exception.
        if (result == null) {
            assertFalse(res.messagesList.isEmpty());
            assertFalse(res.isSuccessfull());
        } else {
            assertNotNull(result);
        }
    }


}
```



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 MySqlLite Workbench 8.0 and the failures do not appear anymore.

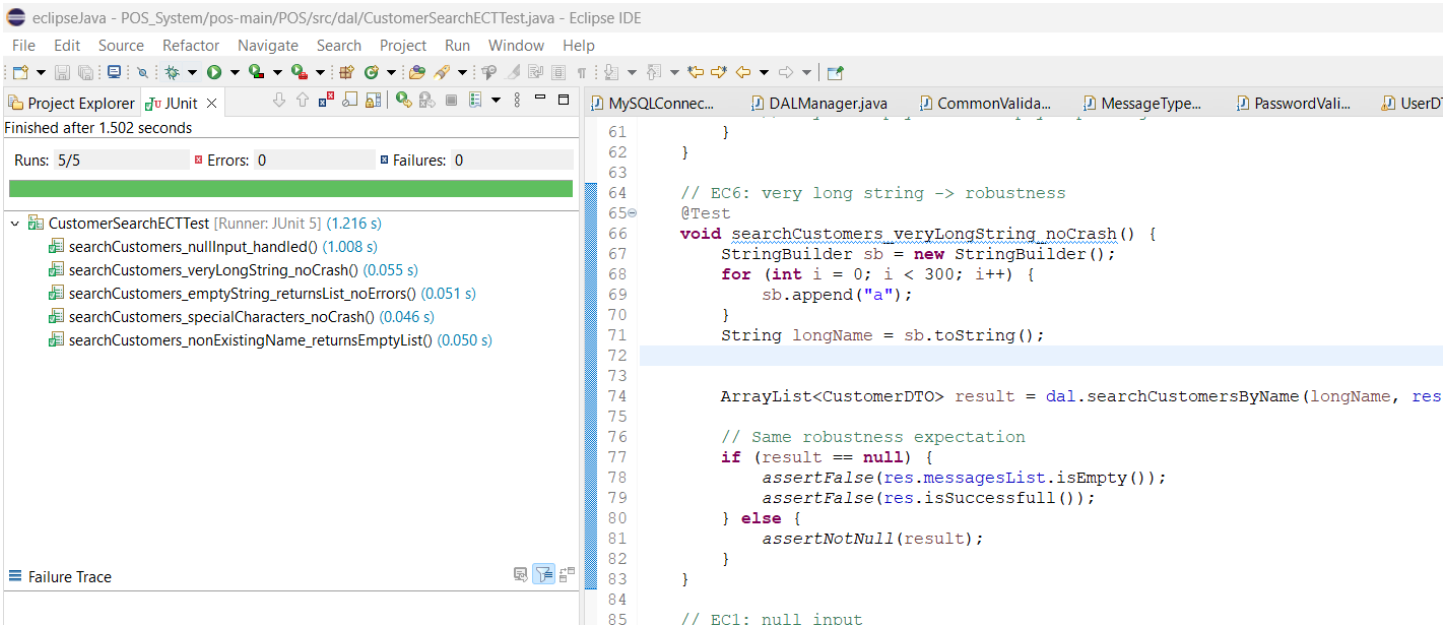
The only error is the repeat() method ->**Java version issue**, not a testing mistake

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(

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

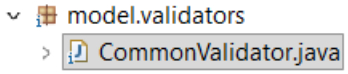


3. **Code Coverage + MC/DC**

Method Under Test:

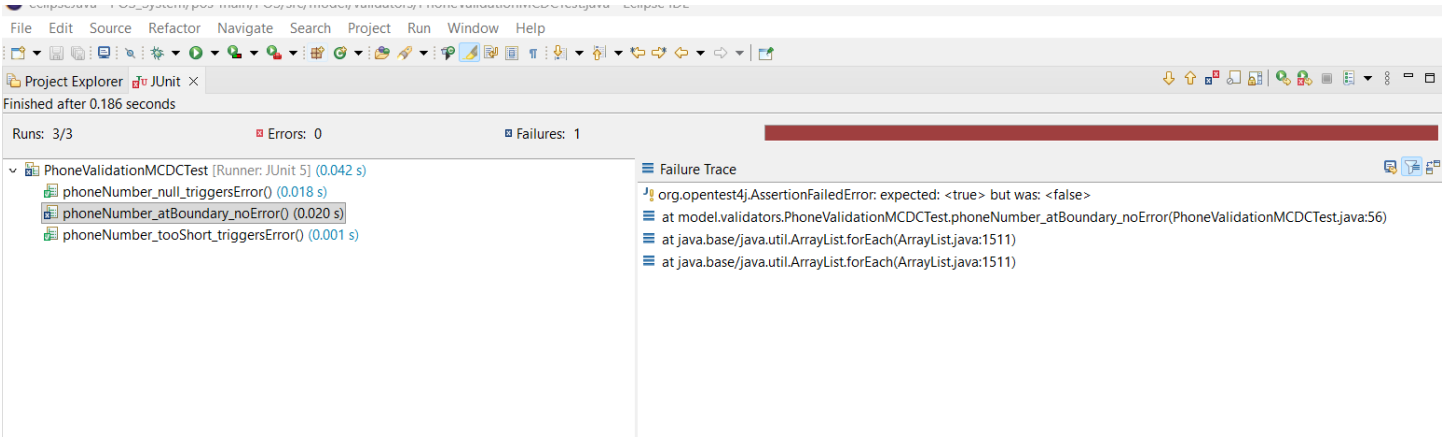
```
   private static void isValildPhoneNo(String phoneNumber, Response response) {
        if (phoneNumber == null || phoneNumber.length() < 10) {
            response.messagesList.add(
                new Message("Phone Number is not valid, provide valid Phone Number with
at least 10 characters.",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

    }
```

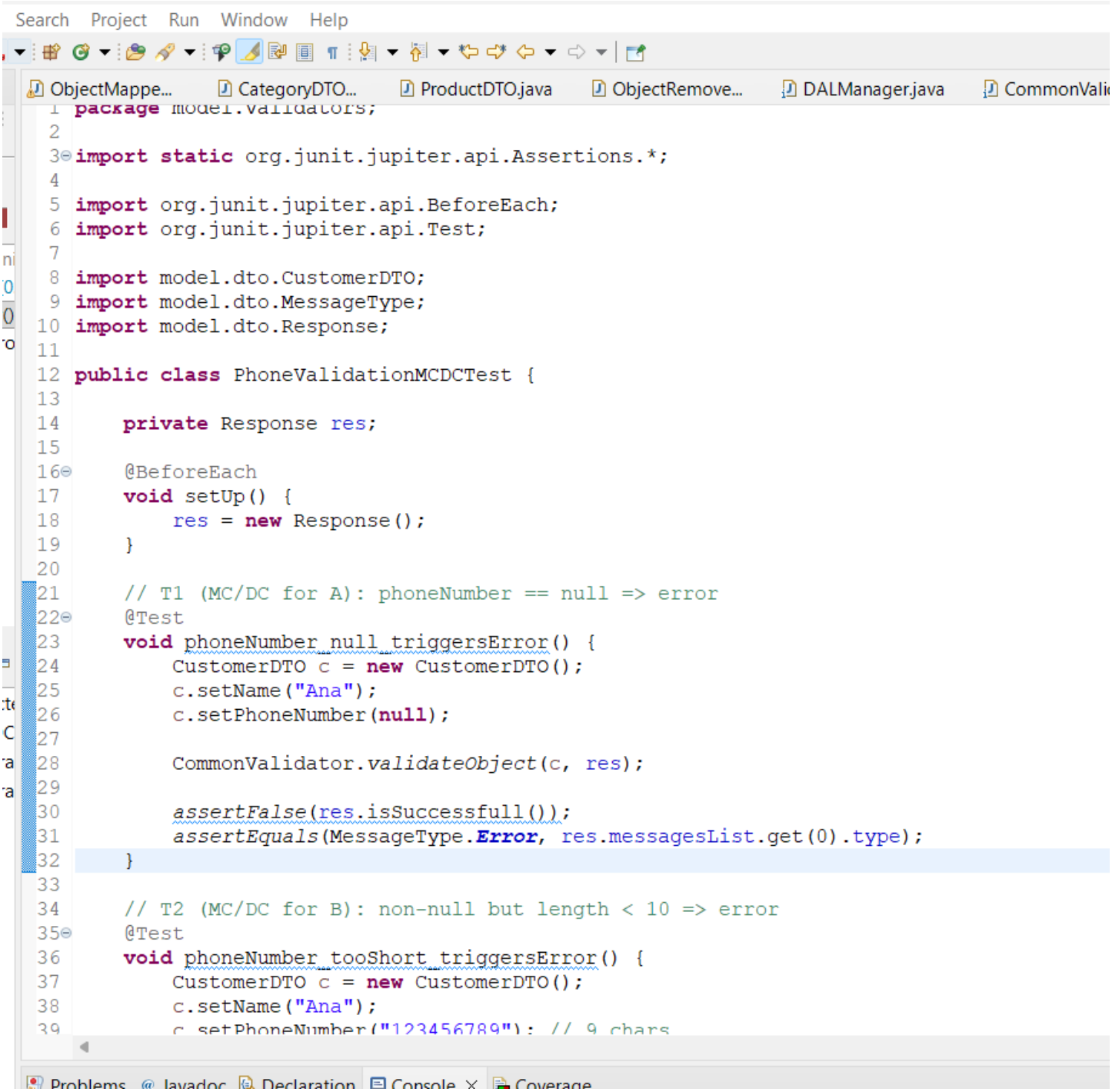
Package and class:



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:




Search Project Run Window Help
ObjectMappe... CategoryDTO... ProductDTO.java ObjectRemove... DALManager.java CommonVali
package model.validators;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import model.dto.CustomerDTO;
import model.dto.MessageType;
import model.dto.Response;
public class PhoneValidationMCDCTest {
private Response res;
@BeforeEach
void setUp() {
res = new Response();
}
// T1 (MC/DC for A): phoneNumber == null => error
@Test
void phoneNumber_null_triggersError() {
CustomerDTO c = new CustomerDTO();
c.setName("Ana");
c.setPhoneNumber(null);
CommonValidator.validateObject(c, res);
assertFalse(res.isSuccessfull());
assertEquals(MessageType.Error, res.messagesList.get(0).type);
}
// T2 (MC/DC for B): non-null but length < 10 => error
@Test
void phoneNumber_tooShort_triggersError() {
CustomerDTO c = new CustomerDTO();
c.setName("Ana");
c.setPhoneNumber("123456789"); // 9 chars
Problems Javadoc Declaration Console Coverage

```
    // T2 (MC/DC for B): non-null but length < 10 => error
    @Test
    void phoneNumber_tooShort_triggersError() {
        CustomerDTO c = new CustomerDTO();
        c.setName("Ana");
        c.setPhoneNumber("123456789"); // 9 chars

        CommonValidator.validateObject(c, res);

        assertFalse(res.isSuccessfull());
        assertEquals(MessageType.Error, res.messagesList.get(0).type);
    }

    // T3 (Both false): length == 10 => no error
    @Test
    void phoneNumber_atBoundary_noError() {
        CustomerDTO c = new CustomerDTO();
        c.setName("Ana");
        c.setPhoneNumber("1234567890"); // 10 chars

        CommonValidator.validateObject(c, res);

        assertTrue(res.isSuccessfull());
    }
}
```

Problems @ Javadoc Declaration Console × Coverage

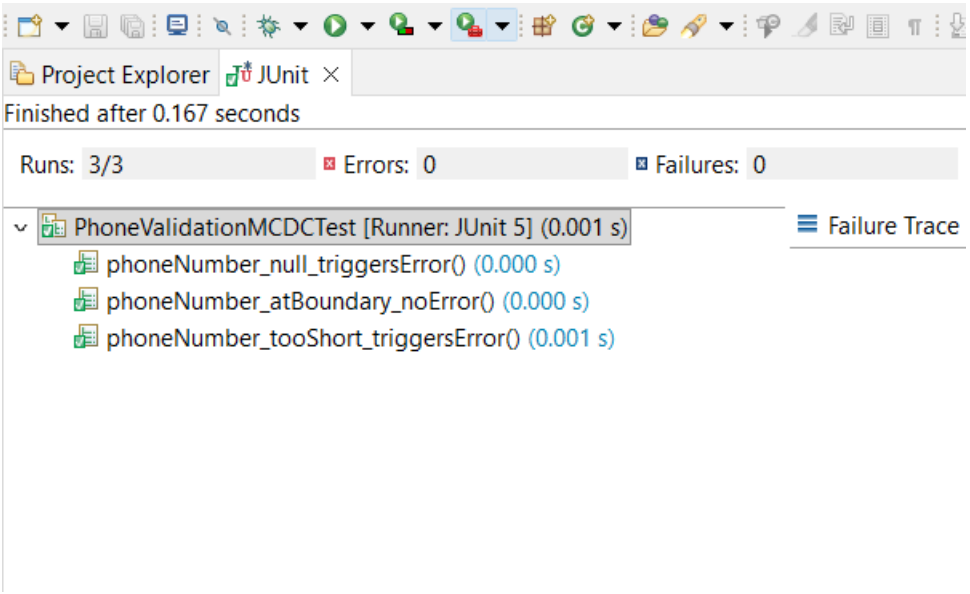
1 failure-> When testing one decision, all other inputs must be valid.
So: Let's give the customer a clearly valid name.
During MC/DC testing of the phone number validation condition, all other input fields were fixed to valid values to ensure that only the phone number conditions independently affected 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


Project Explorer
JUnit
Finished after 0.167 seconds
Runs: 3/3
Errors: 0
Failures: 0
PhoneValidationMCDCTest [Runner: JUnit 5] (0.001 s)
phoneNumber_null_triggersError() (0.000 s)
phoneNumber_atBoundary_noError() (0.000 s)
phoneNumber_tooShort_triggersError() (0.001 s)
Failure Trace