**SE 318**

**SOFTWARE VERIFICATION AND VALIDATION**

**SPRING 2020**

EMPLOYEE TIME TRACKING SYSTEM

GÜN ULUUTKU

**MUSTAFA CAN BAĞDİKEN**

**CEM ÖZCAN**

**BERK KOCAMAN**

UNIT TEST DOCUMENT

Version *<3.0>*

*<05/21/2020>*

VERSION HISTORY

|  |
| --- |
| **VERSION 1.0 (23/04/2020)** |
| Created some of classes which are;  adminOperations, database, EmployeeOperations, main, ManagerOperations and users. Connected to database system. Registration menu added which are name, surname, age, email, username, password and Tc-number. Also added to user authentication by username and password. User can login to system after registration and data kept in databases. |
| **VERSION 2.0 (07/05/2020)** |
| We added more comments to understand code easier. Added a feature to make it possible for manager to approve worksheets. Employees can add and check worksheets. Admin can add, read, update and delete users. Refactored every class in the project. OOP principles applied. Admin, manager and users classes merged to single users class. Main Menu navigation structure has been improved. Negative and positive test cases created. |
| **VERSION 3.0 (21/05/2020)** |
| In this project, negative and positive test cases added. Test suite created. Some problems are fixed. All requirements are completed. |

# INTRODUCTION

## PURPOSE OF THE TEST CASE DOCUMENT

In this document, we write what changed with each version, the programming language and unit test framework work we used for the project and what each test case does.

## CONSTRAINTS

In this project, we used Java as a programming language and JUnit as a unit test framework. Also, Heidi was used as a database in the project.

# UNIT TEST FRAMEWORK: *JUNIT*

In this project, we used JUnit. JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development and is one of a family of unit testing frameworks. A JUnit test is a method contained in a class which is only used for testing. This is called a Test class. To define that a certain method is a test method, annotate it with the @Test annotation.

# TEST CASES

|  |  |
| --- | --- |
| **Test Case 1** | |
| **Test Definition** | |
| **Scenario: Delete a user that does not exists.** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Fail** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void deleteByTC\_Test\_Positive() throws SQLException {**  **Users users = new Users();**  **tf.createDummyRowToUsers();**  **int oldAmount = Integer.parseInt(tf.getLastRowInsertedOnUsers().get("id").toString());**  **users.deleteByTC("12345678901");**  **int newAmount = Integer.parseInt(tf.getLastRowInsertedOnUsers().get("id").toString());**  **Assert.assertTrue(oldAmount!=newAmount);**  **}** |  |

|  |  |
| --- | --- |
| **Test Case 2** | |
| **Test Definition** | |
| **Scenario: Employee logins to the system as an employee** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Success** | **Success** |
| **Result of Test Case** | **Successful** |
| **Test Script** |  |
| **@Test**  **public void login\_As\_Employee\_and\_CheckForAuthgroupPositive() throws SQLException {**  **users.login("e","e");**  **int authgroup = users.\_authgroup;**  **// Test: authgroup is 1**  **Assert.assertEquals(1, authgroup);**  **}** |  |
| **Test Case 3** | |
| **Test Definition** | |
| **Scenario: Manager logins to the system as a manager** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Success** | **Success** |
| **Result of Test Case** | **Successful** |
| **Test Script** |  |
| **@Test**  **public void login\_As\_Manager\_and\_CheckForAuthgroupPositive() throws SQLException {**  **users.login("m","m");**  **int authgroup = users.\_authgroup;**  **// Test: authgroup is 2**  **Assert.assertEquals(2,authgroup);**  **}** |  |

|  |  |
| --- | --- |
| **Test Case 4** | |
| **Test Definition** | |
| **Scenario: Admin logins to the system as an admin** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Success** | **Success** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void login\_As\_Admin\_and\_CheckForAuthgroupPositive() throws SQLException {**  **users.login("123","123");**  **int authgroup = users.\_authgroup;**  **// Test: authgroup is 3**  **Assert.assertEquals(3,authgroup);**  **}** |  |
| **Test Case 5** | |
| **Test Definition** | |
| **Scenario: Manager logins to the system as an employee** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Fail** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void login\_As\_Manager\_and\_CheckForAuthgroupNegative1() throws SQLException {**  **users.login("m","m");**  **int authgroup = users.\_authgroup;**  **// False values to check if they are coming or not**  **Assert.assertNotSame(1,authgroup);**  **}** |  |

|  |  |
| --- | --- |
| **Test Case 6** | |
| **Test Definition** | |
| **Scenario: Manager logins to the system as an admin** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Fail** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void login\_As\_Manager\_and\_CheckForAuthgroupNegative2() throws SQLException {**  **users.login("m","m");**  **int authgroup = users.\_authgroup;**  **// False values to check if they are coming or not**  **Assert.assertNotSame(3,authgroup);**  **}** |  |
| **Test Case 7** | |
| **Test Definition** | |
| **Scenario: Admin logins to the as a manager** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Fail** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void login\_As\_Admin\_and\_CheckForAuthgroupNegative1() throws SQLException {**  **users.login("123","123");**  **int authgroup = users.\_authgroup;**  **// False values to check if they are coming or not**  **Assert.assertNotSame(2,authgroup);**  **}** |  |

|  |  |  |
| --- | --- | --- |
| **Test Case 8** | | |
| **Test Definition** | | |
| **Scenario: Admin logins to the system as an employee** | | |
| **Input Value** | | |
| **<Write input>** | | |
| **Expected Value** | **Actual Value** | |
| **Fail** | **Fail** | |
| **Result of Test Case** | ***Successful*** | |
| **Test Script** |  | |
| **@Test**  **public void login\_As\_Admin\_and\_CheckForAuthgroupNegative2() throws SQLException {**  **users.login("123","123");**  **int authgroup = users.\_authgroup;**  **// False values to check if they are coming or not**  **Assert.assertNotSame(1,authgroup);**  **}** |  | |
| **Test Case 9** | | |
| **Test Definition** | | |
| **Scenario: Employee logins to the system as an admin** | | |
| **Input Value** | | |
| **<Write input>** | | |
| **Expected Value** | | **Actual Value** |
| **Fail** | | **Fail** |
| **Result of Test Case** | | ***Successful*** |
| **Test Script** | |  |
| **@Test**  **public void login\_As\_Employee\_and\_CheckForAuthgroupNegative1() throws SQLException {**  **users.login("e","e");**  **int authgroup = users.\_authgroup;**  **// False values to check if they are coming or not**  **Assert.assertNotSame(3,authgroup);**  **}** | |  |

|  |  |
| --- | --- |
| **Test Case 10** | |
| **Test Definition** | |
| **Scenario: Employee logins to the system as a manager** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Fail** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void login\_As\_Employee\_and\_CheckForAuthgroupNegative2() throws SQLException {**  **users.login("e","e");**  **int authgroup = users.\_authgroup;**  **// False values to check if they are coming or not**  **Assert.assertNotSame(2,authgroup);**  **}** |  |
| **Test Case 11** | |
| **Test Definition** | |
| **Scenario: User logins with wrong username and password** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Fail** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void loginTestNegative() throws SQLException {**  **boolean falseLogin = users.login("qwe","qwe");**  **// Test: falseLogin is true**  **Assert.assertEquals(false, falseLogin);**  **}** |  |

|  |  |
| --- | --- |
| **Test Case 12** | |
| **Test Definition** | |
| **Scenario: Tried to register a user that already exist** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Fail** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void registerTestNegative() {**  **//Connection to database**  **Connection conn = null;**  **Statement st = null;**  **PreparedStatement preparedStatement = null;**  **ResultSet results = null;**  **String negativeTest = "Adnan";**  **String name = null;**  **try{**  **//Connection to database continious**  **conn = db.connect();**  **st = conn.createStatement();**  **preparedStatement = conn.prepareStatement("INSERT INTO users ( authgroup, name, surname, username, password, age, email, tc) " + "VALUES ( ?,? , ?, ? , ?, ?, ?, ?)");**  **// Set all the missing values in the query**  **preparedStatement.setInt(1, 5);**  **preparedStatement.setString(2, negativeTest);**  **preparedStatement.setString(3, negativeTest);**  **preparedStatement.setString(4, negativeTest);**  **preparedStatement.setString(5, negativeTest);**  **preparedStatement.setInt(6, 9999);**  **preparedStatement.setString(7, negativeTest);**  **preparedStatement.setInt(8, 000);**  **//Execute query**  **preparedStatement.execute();**  **String query = "SELECT \* FROM users WHERE authgroup = 3";**  **results = st.executeQuery(query);**  **while(results.next()){**  **name = results.getString("Adnan");**  **}**  **} catch (Exception e){**  **// Test: if negativeTest and name is equal**  **Assert.assertFalse(negativeTest.equals(name));**  **}**  **}** |  |
| **Test Case 13** | |
| **Test Definition** | |
| **Scenario: User registers to the system with high age using TC number** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **<Write Fail result>** | **Fail** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| @Test  public void registerMethodTestNegativeForTC() throws SQLException {  Users user = new Users();  user.register(3,  "denemename",  "denemesurname",  "denemeusername",  "denemeparola",  999999,  "deneme@mail.com",  "12345678901" );  Connection connection = db.connect();  Statement st = connection.createStatement();  String query = "SELECT \* FROM users order by id desc limit 1";  ResultSet results = st.executeQuery(query);  results.next();  Assert.assertNotSame(999999999,results.getInt("age"));  } |  |

|  |  |
| --- | --- |
| **Test Case 14** | |
| **Test Definition** | |
| **Scenario: User registers to the system with high age using database** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Fail** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void registerMethodTestNegativeForHighAgeWithDB() throws SQLException {**  **Users user = new Users();**  **user.register(3,**  **"denemename",**  **"denemesurname",**  **"denemeusername",**  **"denemeparola",**  **999999,**  **"deneme@mail.com",**  **"12345678901" );**  **Connection connection = db.connect();**  **Statement st = connection.createStatement();**  **String query = "SELECT \* FROM users order by id desc limit 1";**  **ResultSet results = st.executeQuery(query);**  **results.next();**  **Assert.assertNotSame(999999999,results.getInt("age"));**  **}** |  |
| **Test Case 15** | |
| **Test Definition** | |
| **Scenario: User registers to the system with underage using control condition** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Fail** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void registerMethodTestNegativeForLowAgeWithControlCondition() throws SQLException {**  **Users user = new Users();**  **boolean result = user.register(3,**  **"denemename",**  **"denemesurname",**  **"denemeusername",**  **"denemeparola",**  **8,**  **"deneme@mail.com",**  **"12345678901" );**  **Assert.assertFalse(result);**  **}** |  |

|  |  |
| --- | --- |
| **Test Case 16** | |
| **Test Definition** | |
| **Scenario: User registers to the system for high age with control condition** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Fail** |
| **Result of Test Case** | ***Success*** |
| **Test Script** |  |
| **@Test**  **public void registerMethodTestNegativeForHighAgeWithControlCondition() throws SQLException {**  **Users user = new Users();**  **boolean result = user.register(3,**  **"denemename",**  **"denemesurname",**  **"denemeusername",**  **"denemeparola",**  **201,**  **"deneme@mail.com",**  **"12345678901" );**  **Assert.assertFalse(result);**  **}** |  |
| **Test Case 17** | |
| **Test Definition** | |
| **Scenario: User registers to the system** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Success** | **Success** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void registerMethodTestPositive() throws SQLException {**  **Users user = new Users();**  **user.register(3,**  **"denemename",**  **"denemesurname",**  **"denemeusername",**  **"denemeparola",**  **99,**  **"deneme@mail.com",**  **"12345678901" );**  **Connection connection = db.connect();**  **Statement st = connection.createStatement();**  **String query = "SELECT \* FROM users order by id desc limit 1";**  **ResultSet results = st.executeQuery(query);**  **results.next();**  **Assert.assertEquals("denemename",results.getString("name"));**  **Assert.assertEquals("denemesurname",results.getString("surname"));**  **Assert.assertEquals("denemeusername",results.getString("username"));**  **Assert.assertEquals("denemeparola",results.getString("password"));**  **Assert.assertEquals(99,results.getInt("age"));**  **Assert.assertEquals("deneme@mail.com",results.getString("email"));**  **Assert.assertEquals("12345678901",results.getString("tc"));**  **}** |  |

|  |  |
| --- | --- |
| **Test Case 18** | |
| **Test Definition** | |
| **Scenario: The registered e-mail doesn’t have any “@”** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Fail** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void registerMethodTestNegativeForMailWithControlStatement1() throws SQLException {**  **// System.out.println(getLastRowInsertedOnUsers().get("id"));**  **Assert.assertFalse(tf.tryMailInput("canbagdiken.com"));**  **}** |  |
| **Test Case 19** | |
| **Test Definition** | |
| **Scenario: The registered e-mail has 2 “@”** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Fail** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void registerMethodTestNegativeForMailWithControlStatement2() throws SQLException {**  **Assert.assertFalse(tf.tryMailInput("can@@bagdiken.com"));**  **}** |  |

|  |  |
| --- | --- |
| **Test Case 20** | |
| **Test Definition** | |
| **Scenario: The registered e-mail doesn’t have any dot** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Fail** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void registerMethodTestNegativeForMailWithControlStatement3() throws SQLException {**  **Assert.assertFalse(tf.tryMailInput("can@bagdikencom"));**  **}** |  |
| **Test Case 21** | |
| **Test Definition** | |
| **Scenario: User logins to the system** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Success** | **Success** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void loginTestPositive() throws SQLException {**  **boolean trueLogin = users.login("e","e");**  **// Test: trueLogin is true**  **Assert.assertEquals(true, trueLogin);**  **}** |  |

|  |  |
| --- | --- |
| **Test Case 22** | |
| **Test Definition** | |
| **Scenario: The number “123” is a positive number** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Success** | **Success** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void isNumericPositive1(){**  **Assert.assertTrue(validationFunctions.isNumeric("123"));**  **}** |  |
| **Test Case 23** | |
| **Test Definition** | |
| **Scenario: The number “0” is a positive number** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Success** |
| **Result of Test Case** | ***Fail*** |
| **Test Script** |  |
| **@Test**  **public void isNumericPositive2(){**  **Assert.assertTrue(validationFunctions.isNumeric("0"));**  **}** |  |

|  |  |
| --- | --- |
| **Test Case 24** | |
| **Test Definition** | |
| **Scenario: The number “-123” is a positive number** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Success** |
| **Result of Test Case** | ***Fail*** |
| **Test Script** |  |
| **@Test**  **public void isNumericPositive3(){**  **Assert.assertTrue(validationFunctions.isNumeric("-123"));**  **}** |  |
| **Test Case 25** | |
| **Test Definition** | |
| **Scenario: The number “abc” is a negative number** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Success** |
| **Result of Test Case** | ***Fail*** |
| **Test Script** |  |
| **@Test**  **public void isNumericNegative2(){**  **Assert.assertFalse(validationFunctions.isNumeric("abc"));**  **}** |  |

|  |  |
| --- | --- |
| **Test Case 26** | |
| **Test Definition** | |
| **Scenario: The number “” is a negative number** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Success** |
| **Result of Test Case** | ***Fail*** |
| **Test Script** |  |
| **@Test**  **public void isNumericNegative3(){**  **Assert.assertFalse(validationFunctions.isNumeric(""));**  **}** |  |
| **Test Case 27** | |
| **Test Definition** | |
| **Scenario: The number “a123b” is a negative number** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Success** |
| **Result of Test Case** | ***Fail*** |
| **Test Script** |  |
| **@Test**  **public void isNumericNegative4(){**  **Assert.assertFalse(validationFunctions.isNumeric("a123b"));**  **}** |  |

|  |  |
| --- | --- |
| **Test Case 28** | |
| **Test Definition** | |
| **Scenario: The number “a123” is a negative number** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Success** |
| **Result of Test Case** | ***Fail*** |
| **Test Script** |  |
| **@Test**  **public void isNumericNegative5(){**  **Assert.assertFalse(validationFunctions.isNumeric("a123"));**  **}** |  |
| **Test Case 29** | |
| **Test Definition** | |
| **Scenario: The number “123a” is a negative number** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **Fail** | **Success** |
| **Result of Test Case** | ***Fail*** |
| **Test Script** |  |
| **@Test**  **public void isNumericNegative6(){**  **Assert.assertFalse(validationFunctions.isNumeric("123a"));**  **}** |  |

|  |  |
| --- | --- |
| **Test Case 30** | |
| **Test Definition** | |
| **Scenario: “can@bagdiken.com” is a valid e-mail** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **True** | **True** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void isValidMailPositive1(){**  **Assert.assertTrue(validationFunctions.isValidMail("can@bagdiken.com"));**  **}** |  |

|  |  |
| --- | --- |
| **Test Case 31** | |
| **Test Definition** | |
| **Scenario: “can.bagdiken@bagdiken.com” is a valid e-mail** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **True** | **True** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void isValidMailPositive2(){**  **Assert.assertTrue(validationFunctions.isValidMail("can.bagdiken@bagdiken.com"));**  **}** |  |

|  |  |
| --- | --- |
| **Test Case 32** | |
| **Test Definition** | |
| **Scenario: “can.bagdiken@std.ieu.edu.tr” is a valid e-mail** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **True** | **True** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void isValidMailPositive4(){**  **Assert.assertTrue(validationFunctions.isValidMail("can.bagdiken@std.ieu.edu.tr"));**  **}** |  |
| **Test Case 33** | |
| **Test Definition** | |
| **Scenario: “can@std.ieu.edu.tr” is a valid e-mail** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **True** | **True** |
| **Result of Test Case** | ***Successful*** |
| **Test Script** |  |
| **@Test**  **public void isValidMailPositive3(){**  **Assert.assertTrue(validationFunctions.isValidMail("can@std.ieu.edu.tr"));**  **}** |  |
| **Test Case 34** | |
| **Test Definition** | |
| **Scenario: “canbagdiken.com” is a valid e-mail** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **True** | **False** |
| **Result of Test Case** | ***Fail*** |
| **Test Script** |  |
| **@Test**  **public void isValidMailNegative1(){**  **Assert.assertFalse(validationFunctions.isValidMail("canbagdiken.com"));**  **}** |  |
| **Test Case 35** | |
| **Test Definition** | |
| **Scenario: “can@bagdiken” is a valid e-mail** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **True** | **False** |
| **Result of Test Case** | ***Fail*** |
| **Test Script** |  |
| **@Test**  **public void isValidMailNegative2(){**  **Assert.assertFalse(validationFunctions.isValidMail("can@bagdiken"));**  **}** |  |

|  |  |
| --- | --- |
| **Test Case 36** | |
| **Test Definition** | |
| **Scenario: “can@” is a valid e-mail** | |
| **Input Value** | |
| **<Write input>** | |
| **Expected Value** | **Actual Value** |
| **True** | **False** |
| **Result of Test Case** | ***Fail*** |
| **Test Script** |  |
| **@Test**  **public void isValidMailNegative3(){**  **Assert.assertFalse(validationFunctions.isValidMail("can@"));**  **}** |  |

**4. CONCLUSION**

**In conclusion, the Employee time tracking system has been done. In this project we used the Java language and unit testing framework which name is JUnit. Lastly, we tested the project.Also we created test scripts.There are 18 negative and 18 positive test cases.Result of test cases, some of them are failed and some of them are successed. In principle, all test cases are expected to be passed.The purpose of creating test cases is to ensure that the implemented program behaves as expected. When test cases fail, the decision whether to release the software depends on the severity of the existing bugs or the number of test cases failed and the number of existing bugs.That’s why some test cases failed in this project.**