**JUNIT 5 with MOCKITO 3**

This is the mostly used combination for testing java applications in real time.

Dev-code, Test-code

One developer will not code the complete application. In a team one developer will code a particular task/Story.

Ex: - EmployeeExcelExport.

Once the coding for the task is completed developer has to confirm if the code is working properly or not. Such process is called as Unit Testing. Here unit means a part we can also call it a story or task.

Unit testing = Testing one part of project developed by Programmer

Unit Testing is done by developer.

Ex: EmployeeRegister Task, Test Employee Register

---------------------------------------------------------------------------------------------------

JUNIT 5: JUNIT is a unit test framework. It is opensource Java library.

=> To test our code(class), we need to define one class that is called as Test case (Test class).

=> To define one Test case, we should use

a. JUNIT 5 Annotations

@Test

@DisplayName

@BeforeEach

@AfterEach

@BeforeAll

@AfterAll

@Disabled

@TestMethodOrder

@RepeatedTest

@Tag

b.JUNIT Asset API(Assertions)

One dependency known as **junit-jupiter-engine** is needed.

Working of Jupiter Engine.

As a Test case we write a class and methods. Objects of the class and calling of methods is done by JUNIT platform.

=> Here, Junit Runtime is provided with 3 components and one platform runtime.

i. Junit Jupiter Engine (JUnit 5 API)

ii. Junit Vintage Engine (Junit 4 and 3 APIs)

iii.Integration (TestNg,Mock ,…...etc)

=> For Our Test cases (class + method) object creation and test-method calling done by Junit Platform Runtime.

Admin module: TestAdmin

save () => testSave()

delete () => testDelete()

Product module: TestProduct

-----Writing Test Case--------------------------------------------

1.create one Maven Project

>File>New > Maven Project> choose checkbox

[v] create simple project >Next>Enter details

group Id: aap

artifact Id: junit-5-basics

version:1.0

>finish

2.Open pom.xml provide java version and JUnit dependencies.

=> after adding content in pom.xml ctrl+A =>ctrl+I => ctrl+S

3.create a test case (class) under folder src/test/java

>Right click on ‘src/test/java’ > new > class > enter details

Name : TestEmployee

Package : aap.test

4. Add one method TODO testing. [Test Method]

Syntax:

@Test

Public void <methodName>() {

// test logic

}

5.After writing test case with test method, run it

>Right click on code>Run as -> JUnit test

-----Example Code#1-------------------------------------------------

package aap;

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

import org.junit.jupiter.api.Test;

//Test case: A class that will test our code

class MathUtilsTest {

//test methods

@Test

void testSave() {

System.out.println("Hello-Save");

}

@Test

void testUpdate() {

System.out.println("Hello-Update");

}

@Test

void testDelete() {

System.out.println("Hello-Delete");

}

}

===========================================================@TestMethodOrder: We can define multiple methods inside Testcase.

Those are executed in Random order by default

We can specify our own order using @TestMethodOrder + OrderAnnotation

Here we need to provide @order(number)

package aap;

import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;

import org.junit.jupiter.api.Order;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestMethodOrder;

//Test case: A class that will test our code

@TestMethodOrder(value = OrderAnnotation.class)

class MathUtilsTest {

//test methods

@Test

@Order (1)

void testSave() {

System.out.println("Hello-Save");

}

@Test

@Order(2)

void testUpdate() {

System.out.println("Hello-Update");

}

@Test

@Order(3)

void testDelete() {

System.out.println("Hello-Delete");

}

}

\*) We can use TestMethodOrder + Alphanumeric. Class to provide test method order

First sort using 0-9 if same found then compare with A-Z sorting order.

package aap;

===========Example#2=======================================

import org.junit.jupiter.api.MethodOrderer.Alphanumeric;

import org.junit.jupiter.api.Order;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestMethodOrder;

//Test case: A class that will test our code

@TestMethodOrder (value = Alphanumeric. Class)

class MathUtilsTest {

//test methods

@Test

void testSave() {

System.out.println("Hello-Save");

}

@Test

void testUpdate() {

System.out.println("Hello-Update");

}

@Test

void testDelete() {

System.out.println("Hello-Delete");

}

}

===========================================================@BeforeEach: To execute any logic once per test method before starting test method.

@AfterEach: To execute any logic once per test method after finishing test method.

@BeforeAll: To execute any logic once per test case before starting

@AfterAll: To execute any logic once per test case after finishing.

=======Example#3============================================ package aap;

import org.junit.jupiter.api.AfterAll;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

//Test case: A class that will test our code

class MathUtilsTest {

@BeforeAll

public static void setupOnce() {

System.out.println("One time setup");

}

@BeforeEach

public void setup () {

//setup, init data

System.out.println("FROM SETUP");

}

@Test

void testSave() {

System.out.println("Hello-Save");

}

@Test

void testUpdate() {

System.out.println("Hello-Update");

}

@AfterEach

public void clear() {

System.out.println("CLEAR DATA");

}

@AfterAll

public static void closeAll() {

System.out.println("Once at End");

}

}

Output:

One time setup

FROM SETUP

Hello-Save

CLEAR DATA

FROM SETUP

Hello-Update

CLEAR DATA

Once At End

\*) @DisplayName : This annotation is used to provide ‘Readable Text’ in place of actual method and class names at Junit console

============Example#4======================================= package aap;

import org.junit.jupiter.api.AfterAll;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Test;

//Test case: A class that will test our code

@DisplayName("Testing Employee Task")

class MathUtilsTest {

@BeforeAll

public static void setupOnce() {

System.out.println("One time setup");

}

@BeforeEach

public void setup() {

//setup, init data

System.out.println("FROM SETUP");

}

@Test

@DisplayName("Testing Save Method")

void testSave() {

System.out.println("Hello-Save");

}

@Test

@DisplayName ("Testing Update Method")

void testUpdate() {

System.out.println("Hello-Update");

}

@AfterEach

public void clear() {

System.out.println("CLEAR DATA");

}

@AfterAll

public static void closeAll() {

System.out.println("Once At End");

}

}

\*) @Disabled : This annotation is used to specify ignore one test-method while executing test-case (do not execute test-method)

============Example#5=======================================package aap;

import org.junit.jupiter.api.Disabled;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Test;

//Test case: A class that will test our code

@DisplayName("Testing Employee Task")

class MathUtilsTest {

@Test

@DisplayName("Testing Save Method")

void testSave() {

System.out.println("Hello-Save");

}

@Test

@DisplayName("Testing Update Method")

@Disabled

void testUpdate() {

System.out.println("Hello-Update");

}

}

---------------------------------------------------------------------------------------------------@RepeatedTest and TestInfo

=>To execute any test method multiple time (like batch processing)

Ex: Export 1-10 records

Export 11-20 records

=>To know test case details like class name, method name, display name, tag name etc we can use one interface TestInfo

==========Example#6========================================= package aap;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.RepeatedTest;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestInfo;

//Test case: A class that will test our code

@DisplayName ("Testing Employee Task")

class MathUtilsTest {

@RepeatedTest (value=3, name = "{displayName}- {currentRepetition}/{totalRepetitions}")

@DisplayName ("Multiple Test")

void testMultiple(TestInfo info) {

// System.out.println("Hello: "+info.getTestClass(). get().getName());

// System.out.println("Hello: "+info.getTestMethod(). get().getName());

System.out.println("Hello: "+info.getDisplayName());

}

}

===========================================================@Tag: These are used to filter test methods for execution in different environment

For example, I want to test export example in production environment at same I want to test delete operation only in development environment then use tag concept and maven-surefire-plugin in pom.xml

===================Example#6================================ package aap;

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

import org.junit.jupiter.api.Tag;

import org.junit.jupiter.api.Test;

//Test case: A class that will test our code

class MathUtilsTest {

@Test

@Tag("dev")

public void testA() {

System.out.println("HELLO-TEST-A");

}

@Test

@Tag("prod")

public void testB() {

System.out.println("HELLO-TEST-B");

}

@Test

@Tag("dev")

public void testC() {

System.out.println("HELLO-TEST-C");

}

@Test

@Tag("prod")

public void testD() {

System.out.println("HELLO-TEST-D");

}

}

====================pom.xml================================

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0-M7</version>

<configuration>

<groups>prod</groups>

<excludedGroups>dev</excludedGroups>

</configuration>

</plugin>

</plugins>

</build>

=> Right Click on Project > Run as > maven test

(or)

=>Right click on code > run as> Run configuration

>click on configuration

> provide Include(dev) and exclude(prod) option > Apply and run

-----------------------------------------------------------------------------------------------

**ASSERT API**

Assert API:

It is used to validate Test, IS CURRENT TEST PASS/FAIL?

Assertion methods compare expected value with actual result

=> JUnit 5 has provided class: Assertions(org.junit.jupiter.api) which contains all static methods “assert methods”

=>assert methods are used to compare Expected values with Actual Results.

If matching TEST PASS, else TEST FAILS

=>Ex assert method: assertEquals(expected, actual)

This method is used to compare expected value with actual value

----------------------------------------Code-----------------------------------------------------

1. Create one maven simple project
2. Pom.xml

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

<junit.jupiter.version>5.4.0</junit.jupiter.version>

</properties>

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<version>${junit.jupiter.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>${junit.jupiter.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

1. Define one class under src/main/java which needs testing

package aap.service;

public class Message {

public String showMsg(String name) {

return "Welcome to Mr/Mrs/Ms :"+name;

}

}

4.Test case under src/test/java

package aap.test;

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

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import aap.service.Message;

public class TestMessage {

//Declare Variables

private Message m;

private String expected;

private String actual;

//Provide Initial Data

@BeforeEach

public void setup () {

m = new Message ();

expected= "Welcome to: ABC";

actual = "";

}

//Do Unit Test

@Test

public void testShowMsg() {

actual= m.showMsg("ABC");

assertEquals(expected,actual,"Data may not be matching");

}

//clear heap data /clear memory

@AfterEach

public void clean () {

m=null;

expected=actual=null;

}

}

\*) assertNotNull()/assertNull()

assertNotNull(Object)

This method is used to specify that given object is not a null value it holds data, else TEST FAIL

assertNull(Object) it indicates given object is null, else TEST FAIL.

\*)asssertDoesNotThrow(Executable):

This is used to specify that our method call is not throwing any exception, else if it is throwing then TEST FAIL

--------------------Code-------------------------------------------------------------------

1. Project code

package aap.util;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DBConnection {

Connection con=null;

public Connection getCon() throws ClassNotFoundException, SQLException {

Class.forName("com.mysql.cj.jdbc.Driver");

DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql","root","root");return con;

}}

2.Unit Test Code

package aap.test;

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

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

import java.sql.Connection;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import aap.util.DBConnection;

public class TestDBConnection {

private DBConnection dbc;

private Connection con;

@BeforeEach

public void initData() {

dbc = new DBConnection ();

}

@Test

public void testGetCon() {

assertDoesNotThrow(()->con = dbc.getCon());

//Excepted conn is not a null

assertNotNull(con,"Connection is not created, Please check");

}

@AfterEach

public void clean () {

dbc=null;

con=null;

}

}

---------------------------------------------------------------------------------------------------\*) assertSame(ob1, ob2): This method is used to test that GIVEN TWO REFERENCES are POINTED TO ONE OBJECT.

If yes TEST PASS, else TEST FAIL

Q) what is the diff b/w assertSame() and assertEquals()?

A) assertEquals(): compares two objects' data

assertSame(): compares two objects' references

\*) fail (): This is used for testing multiple conditions, if they are not met manually FAIL TEST CASE

---------------------------Example: Testing Singleton DB Connections------------------1.pom.xml

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

<junit.jupiter.version>5.4.0</junit.jupiter.version>

</properties>

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<version>${junit.jupiter.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>${junit.jupiter.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.29</version>

</dependency>

</dependencies>

2.Project Code

package aap.util;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DBConnectionUtil {

private static Connection con;

static {

try {

Class.forName("com.mysql.cj.jdbc.Driver");

} catch (ClassNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

try {

DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql","root","root"); } catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

public static Connection getCon() {

return con;

}

}

3.Test Case

package aap.test;

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

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

import java.sql.Connection;

import org.junit.jupiter.api.Test;

import aap.util.DBConnectionUtil;

public class TestDBConnectionUtil {

@Test

public void testGetCon() {

Connection con1 = DBConnectionUtil.getCon();

Connection con2 = DBConnectionUtil.getCon();

//assertNotNull(con1,"Connection may be created...");

if (con1==null|| con2==null) {

//TEST CASE FAILED

fail ("CONNECTIONS ARE NOT CREATED");

}

assertSame(con1, con2,"May not be same connection");

}

}

---------------------------------------------------------------------------------------------

\*) assertArraysEquals(): use this method to compare data of two arrays (expected, actual)

Test case:

package aap.test;

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

import org.junit.jupiter.api.Test;

public class TestSample {

@Test

public void testNormal() {

int [] expected = {10,20,30};

int [] actual = {10,20};

assertArrayEquals(expected,actual,"Data may not be same in two arrays!");

}

}

\*)assertTrue()/assertFalse()

These methods are used to test one boolean condition/expression/value.

assertTrue(): boolean value is expected as TRUE,else TEST FAIL

assertFalse(): boolean value is expected as FALSE,else TEST FAIL

--------Example----------------------

package aap.test;

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

import org.junit.jupiter.api.Test;

public class TestSample {

@Test

public void testNormal() {

boolean exists= false;

assertTrue(exists,"Data may not exists!");

}

}

\*) assertThrows(): expecting that our logic throws one exception as: T (Type)

assertThrows(ExpectedExceptionType.class,()-> {our logic});

---Example----

package aap.test;

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

import org.junit.jupiter.api.Test;

public class TestSample {

@Test

public void testNormal() {

assertTrue(exists,"Data may not exists!");

assertThrows(NullPointerException.class,()-> {throw new ArrayIndexOutOfBoundsException ();});

}

}

---------------------------------------------------------------------------------------------------

assertAll(): This is used to group all assert test methods and execute once.

If all are PASS, then TEST is PASS, if one FAIL then TEST is FAIL

-----------------Example-----------------------------------------

package aap.test;

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

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

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

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

import java.sql.Connection;

import org.junit.jupiter.api.Test;

import aap.util.DBConnectionUtil;

public class TestSample {

@Test

public void testNormal() {

assertAll(()-> {

assertNotNull(DBConnectionUtil.getCon());

},

()->{

Connection con1, con2;

con1=DBConnectionUtil.getCon();

con2=DBConnectionUtil.getCon();

assertSame(con1, con2);

},

()->{

Connection con1, con2;

con1=DBConnectionUtil.getCon();

con2=DBConnectionUtil.getCon();

if(con1==null&&con2==null)

{

fail ();

}

});

}

}

\*) NOTE:

Executable is a functional interface that is having abstract method

void execute () throws Throwable;

So, lambda expression looks like: () -> {}