**WEEK – 2(Hands-on)**

**TDD using JUnit5 and Mockito**

**JUnit\_Basic Testing Exercises**

**Exercise 1 : Setting Up**

**App.java**

package com.springmvc.MavenApplication;

public class App {

public static void main(String[] args) {

System.*out*.println("Hello World!");

}

}

**AppTest.java**

package com.springmvc.MavenApplication;

import org.junit.jupiter.api.Test;

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

public class AppTest {

*@Test*

public void shouldAnswerWithTrue() {

// Assert true

*assertTrue*(5 > 3);

}

}

**Exercise 3: Assertions in Junit**

**Program:**

import org.junit.jupiter.api.Test;

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

public class AppTest {

*@Test*

public void shouldAnswerWithTrue() {

*assertEquals*(5, 2 + 3);

// Assert true

*assertTrue*(5 > 3);

// Assert false

*assertFalse*(5 < 3);

// Assert null

*assertNull*(null);

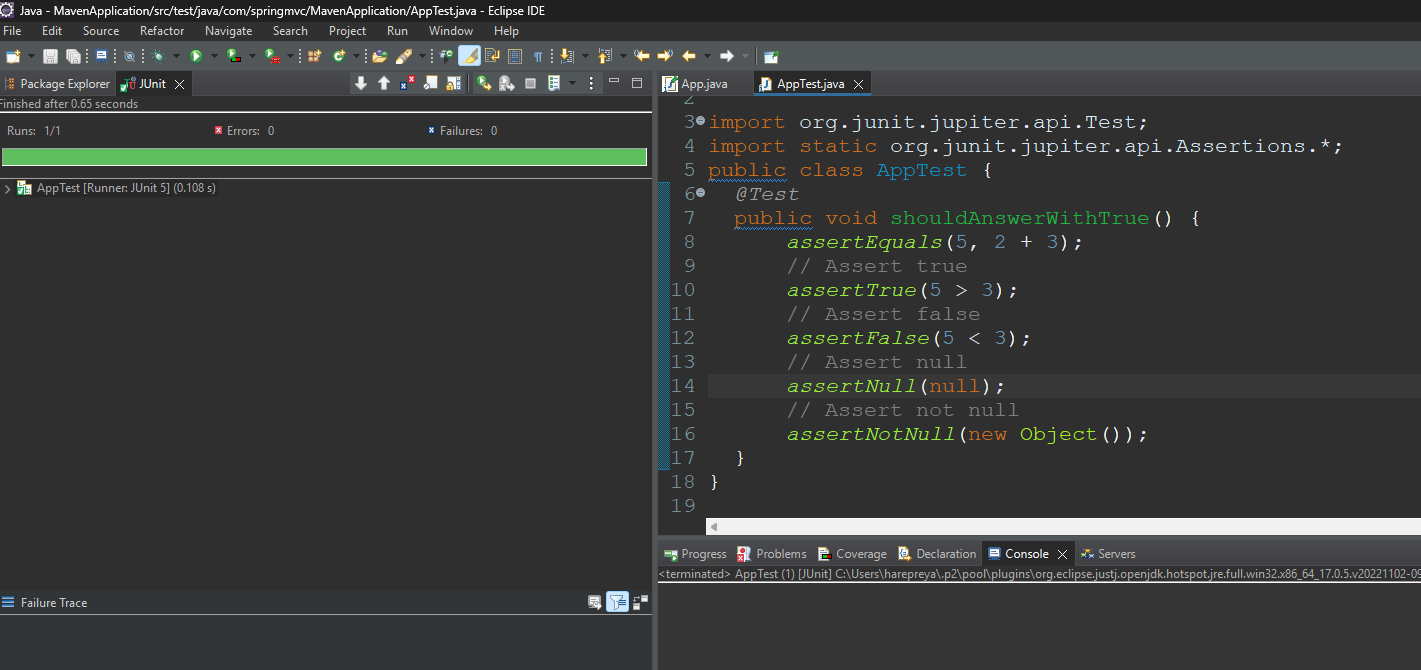
// Assert not null

*assertNotNull*(new Object());

}

}

**Output:**

****

**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in Junit**

**Program:**

**Calculator.java**

package com.springmvc.MavenApplication;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

public int divide(int a, int b) {

if (b == 0) throw new ArithmeticException("Cannot divide by zero");

return a / b;

}

}

**AppTest.java**

package com.springmvc.MavenApplication;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AppTest {

private Calculator calculator;

// Setup before each test

@Before

public void setUp() {

calculator = new Calculator();

System.out.println("Setup complete");

}

// Teardown after each test

@After

public void tearDown() {

calculator = null;

System.out.println("Teardown complete");

}

@Test

public void testAddition() {

// Arrange done in setUp()

// Act

int result = calculator.add(2, 3);

// Assert

assertEquals(5, result);

}

@Test

public void testMultiplication() {

int result = calculator.multiply(4, 5);

assertEquals(20, result);

}

@Test

public void testDivision() {

int result = calculator.divide(10, 2);

assertEquals(5, result);

}

@Test(expected = ArithmeticException.class)

public void testDivisionByZero() {

calculator.divide(10, 0);

}}

**Mockito exercises**

**Exercise 1 :** **Mocking and Stubbing**

**Program:**

**MyService.java**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

// External API interface that the service depends on

interface ExternalApi {

String getData();

}

// The service that uses the ExternalApi

class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api; }

public String fetchData() {

return api.getData();

}}

**MyServiceTest.java**

// Test class using Mockito

public class MyServiceTest {

@Test

public void testExternalApi() {

// Arrange: create mock object

ExternalApi mockApi = mock(ExternalApi.class);

// Stub method to return mock value

when(mockApi.getData()).thenReturn("Mock Data");

// Act: pass mock to service

MyService service = new MyService(mockApi);

String result = service.fetchData();

// Assert: check expected result

assertEquals("Mock Data", result);

}

}

**Pom.xml**

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.7.0</version>

<scope>test</scope>

</dependency>

</dependencies>

**Exercise 2 : Verifying Interactions**

**Program:**

**MyService.java**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

// Simulated external API interface

interface ExternalApi {

String getData();

}

// Service class that uses the external API

class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}}

**MyServiceTest.java**

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Arrange

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

// Act

service.fetchData();

// Assert - verify that getData() was called on the mock

verify(mockApi).getData();

}}

**Pom.xml**

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.7.0</version>

<scope>test</scope>

</dependency>

</dependencies>