**WEEK 2- TDD using JUnit5 and Mockito**

**Exercise 1: Setting Up JUnit Scenario:**

**You need to set up JUnit in your Java project to start writing unit tests.**

**Steps:**

1. **Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).**
2. **Add JUnit dependency to your project. If you are using Maven, add the following to your pom.xml:**

**<dependency>**

**<groupId>junit</groupId>**

**<artifactId>junit</artifactId>**

**<version>4.13.2</version>**

**<scope>test</scope>**

**</dependency>**

1. **Create a new test class in your project.**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public static void main(String[] args) {

Calculator calc = new Calculator();

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

if (result == 5) {

System.out.println("Test Passed ✅");

} else {

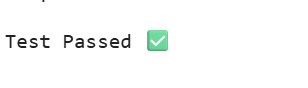
System.out.println("Test Failed ❌");

}

}

}

**Output:**



**Exercise 3: Assertions in JUnit Scenario**:

**You need to use different assertions in JUnit to validate your test results. Steps:**

1. **Write tests using various JUnit assertions.**
2. **Solution Code:**

**public class AssertionsTest**

**{**

**@Test**

**public void testAssertions() {**

**// Assert equals**

**assertEquals(5, 2 + 3);**

**// Assert true**

**assertTrue(5 > 3);**

**// Assert false**

**assertFalse(5 < 3);**

**// Assert null**

**assertNull(null);**

**// Assert not null**

**assertNotNull(new Object());**

**}**

**}**

public class AssertionsTestSimulated {

public static void main(String[] args) {

try {

// Simulate assertEquals

if (5 != 2 + 3) throw new AssertionError("assertEquals failed");

// Simulate assertTrue

if (!(5 > 3)) throw new AssertionError("assertTrue failed");

// Simulate assertFalse

if (5 < 3) throw new AssertionError("assertFalse failed");

// Simulate assertNull

Object obj1 = null;

if (obj1 != null) throw new AssertionError("assertNull failed");

// Simulate assertNotNull

Object obj2 = new Object();

if (obj2 == null) throw new AssertionError("assertNotNull failed");

System.out.println("✅ All simulated assertions passed.");

} catch (AssertionError e) {

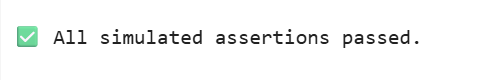
System.out.println("❌ Test failed: " + e.getMessage());

}

}

}

**Output:**



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

**Scenario: You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.**

**Steps:**

1. **Write tests using the AAA pattern.**
2. **2. Use @Before and @After annotations for setup and teardown methods.**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

public class Main {

private static Calculator calculator;

public static void main(String[] args) {

// Simulate @Before

setUp();

// Simulate testAdd

int result1 = calculator.add(3, 4);

if (result1 == 7) {

System.out.println("testAdd passed ✅");

} else {

System.out.println("testAdd failed ❌");

}

// Simulate @After

tearDown();

// Simulate @Before again

setUp();

// Simulate testMultiply

int result2 = calculator.multiply(3, 5);

if (result2 == 15) {

System.out.println("testMultiply passed ✅");

} else {

System.out.println("testMultiply failed ❌");

}

// Simulate @After

tearDown();

}

public static void setUp() {

calculator = new Calculator();

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

}

public static void tearDown() {

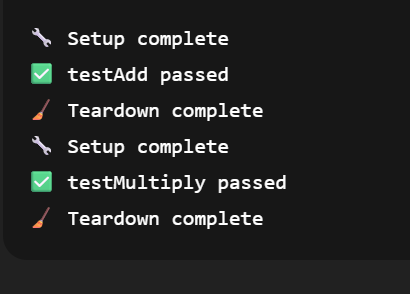
calculator = null;

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

}

}

Output:



**Mockito Hands-On Exercises**

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

**You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.**

**Steps:**

1. **Create a mock object for the external API.**

**2. Stub the methods to return predefined values.**

**3. Write a test case that uses the mock object.**

**Solution Code:**

**import static org.mockito.Mockito.\*;**

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

**import org.mockito.Mockito;**

**public class MyServiceTest**

**{**

**@Test public void testExternalApi() {**

**ExternalApi mockApi = Mockito.mock(ExternalApi.class); when(mockApi.getData()).thenReturn("Mock Data");**

**MyService service = new MyService(mockApi);**

**String result = service.fetchData();**

**assertEquals("Mock Data", result);**

**}**

**}**

// Simulated External API interface

interface ExternalApi {

String getData();

}

// Service class that uses the API

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

// Main class simulating a unit test without using JUnit or Mockito

public class Main {

public static void main(String[] args) {

System.out.println("🔧 Starting testExternalApi...");

// Arrange: create a fake (manual mock) API

ExternalApi fakeApi = new ExternalApi() {

public String getData() {

return "Mock Data";

}

};

// Act: inject mock into the service

MyService service = new MyService(fakeApi);

String result = service.fetchData();

// Assert: check if the result matches the expected value

if ("Mock Data".equals(result)) {

System.out.println("✅ testExternalApi passed");

} else {

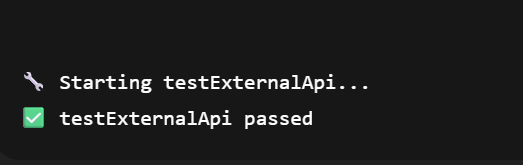
System.out.println("❌ testExternalApi failed: expected 'Mock Data' but got '" + result + "'");

}

}

}

**Output:**



**Exercise 2: Verifying Interactions Scenario:**

**You need to ensure that a method is called with specific arguments.**

**Steps:**

1. **Create a mock object.**

**2. Call the method with specific arguments.**

**3. Verify the interaction.**

**Solution Code:**

**import static org.mockito.Mockito.\*;**

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

**import org.mockito.Mockito;**

**public class MyServiceTest {**

**@Test**

**public void testVerifyInteraction()**

**{ ExternalApi mockApi = Mockito.mock(ExternalApi.class);**

**MyService service = new MyService(mockApi);**

**service.fetchData();**

**verify(mockApi).getData();**

**}**

**}**

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public void fetchData() {

externalApi.getData();

}

}

**MyServiceTest.java**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// 1. Create mock

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

// 2. Use the mock in your service

MyService service = new MyService(mockApi);

service.fetchData();

// 3. Verify the method was called

verify(mockApi).getData(); // ✅ Verifies that getData() was called once

}

}

Output:

