TDD using JUnit5 and Mockito

***1. JUnit\_Basic Testing Exercises***

**Exercise 1: Setting Up JUnit**

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</ground>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

3. Create a new test class in your project.

**Code:**

SimpleMath.java

public class SimpleMath {

public int add(int a, int b) {

return a + b;

}

}

SimpleMathTest.java

import org.junit.jupiter.api.Test;

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

public class SimpleMathTest {

@Test

public void testAdd() {

SimpleMath math = new SimpleMath();

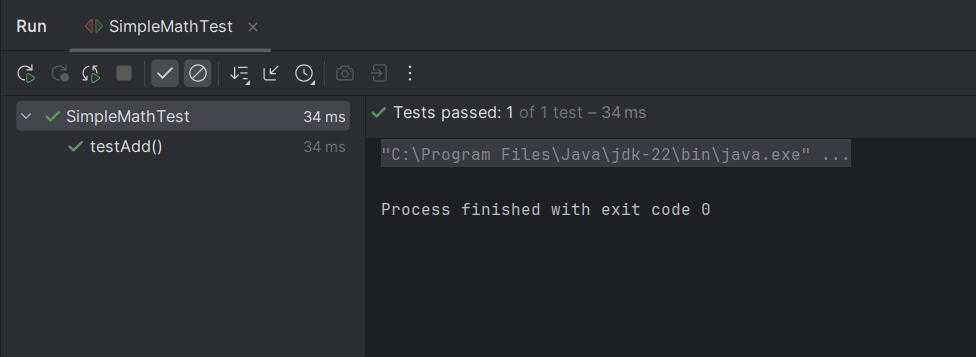
int result = math.add(2, 3);

assertEquals(5, result);

}

}

**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.

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());

}

}

**Code:**

AssertionsTest.java

package com.example.junit;

import org.junit.jupiter.api.Test;

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

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

assertFalse(5 < 3);

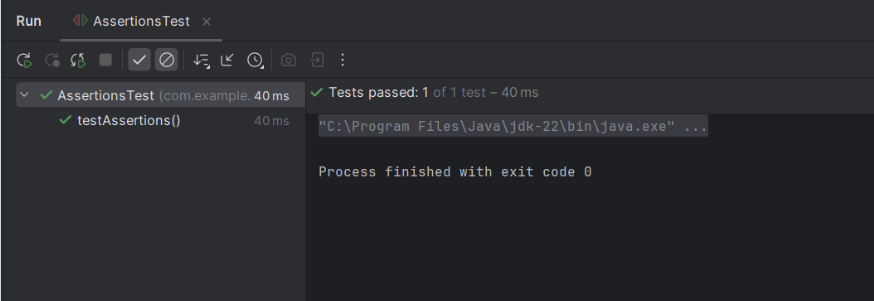
assertNull(null);

assertNotNull(new Object());

}

}

**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. Use @Before and @After annotations for setup and teardown methods.

**Code:**

Calculator.java

package com.example.junit;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

CalculatorTest.java

package com.example.junit;

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

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

public class CalculatorTest {

private Calculator calculator;

@BeforeEach

public void setUp() {

calculator = new Calculator();

System.out.println("Setup: Calculator object created");

}

@AfterEach

public void tearDown() {

System.out.println("Teardown: Test finished");

}

@Test

public void testAddition() {

int result = calculator.add(10, 5);

assertEquals(15, result);

}

@Test

public void testSubtraction() {

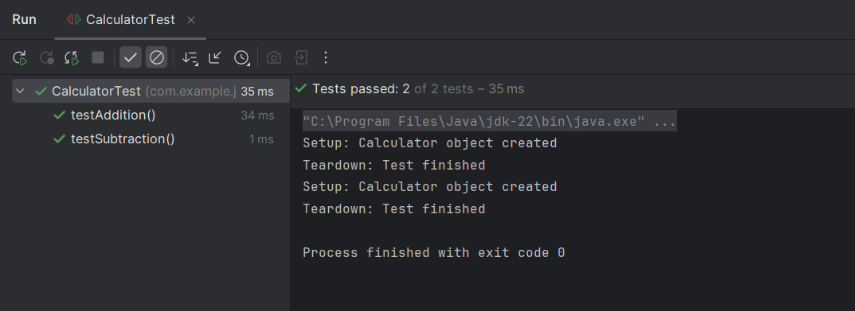
int result = calculator.subtract(10, 4);

assertEquals(6, result);

}

}

**Output:**



***2. Mockito 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);

}

}

**Code:**

ExternalApi.java

package com.example.mockito;

public interface ExternalApi {

String getData();

}

MyService.java

package com.example.mockito;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

MyServiceTest.java

package com.example.mockito;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

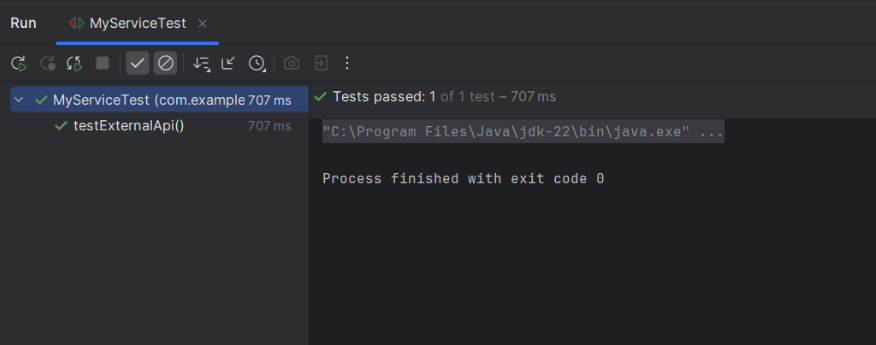
String result = service.fetchData();

assertEquals("Mock Data", 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();

}

}

**Code:**

ExternalApi.java

package com.example.mockito.mockito;

public interface ExternalApi {

String getData();

}

MyService.java

package com.example.mockito.mockito;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api){

this.api = api;

}

public String fetchData(){

return api.getData();

}

}

MyServiceTest.java

package com.example.mockito;

import com.example.mockito.mockito.ExternalApi;

import com.example.mockito.mockito.MyService;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.when;

public class MyServiceTest {

@Test

public void testVerifyInteraction(){

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**Output:**

