Hands-on 2:

TDD using JUnit5 and Mockito

Exercise 1: Setting Up Junit

File name: pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>Calculator</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<version>3.1.2</version>

</plugin>

</plugins>

</build>

</project>

Exercise 3: Assertions in Junit

File name : Calculator.java

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public double divide(int a, int b) {

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

return (double) a / b;

}

public boolean isPositive(int num) {

return num > 0;

}

public Integer getNullValue() {

return null;

}

}

File name : CalculatorTest.java

package com.example;

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

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

public class CalculatorTest {

private Calculator calculator;

@BeforeEach

void setUp() {

calculator = new Calculator();

}

@AfterEach

void tearDown() {

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

}

@Test

void testAddition() {

assertEquals(9, calculator.add(4, 5));

}

@Test

void testDivision() {

assertEquals(2.5, calculator.divide(5, 2), 0.001);

}

@Test

void testDivisionByZero() {

assertThrows(IllegalArgumentException.class, () -> calculator.divide(10, 0));

}

@Test

void testIsPositive() {

assertTrue(calculator.isPositive(7));

assertFalse(calculator.isPositive(-1));

}

@Test

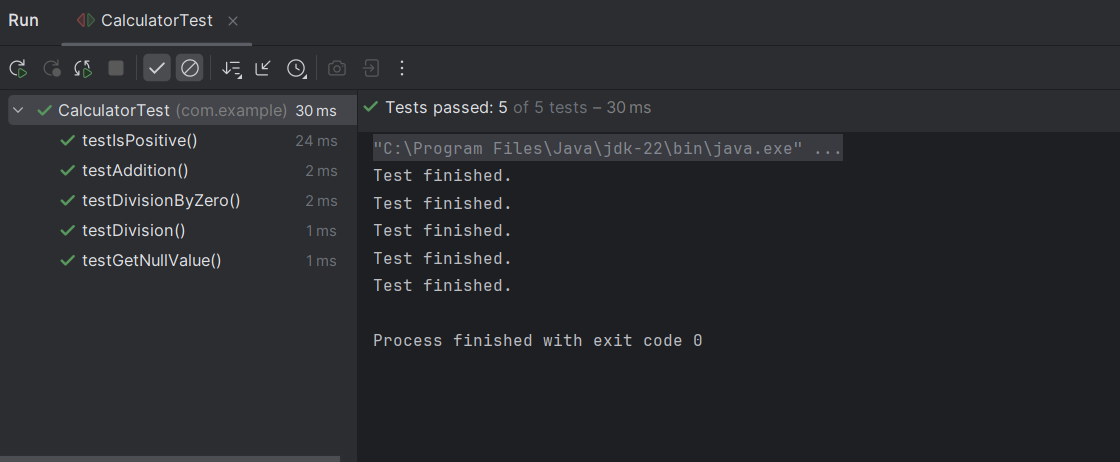
void testGetNullValue() {

assertNull(calculator.getNullValue());

}

}

OUTPUT:



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

File name : CalculatorTest.java

package com.example;

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

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

public class CalculatorTest {

private Calculator calculator;

@BeforeAll

public static void init(){

System.out.println("Calculator testing started");

}

@AfterAll

public static void cleanAll(){

System.out.println("All test completed");

}

@BeforeEach

void setUp() {

calculator = new Calculator();

}

@AfterEach

void tearDown() {

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

}

@Test

void testAddition() {

//Arrange

int a =10;

int b=20;

//Act

int result=calculator.add(a , b);

//Assert

assertEquals(30 , result);

}

@Test

void testDivision() {

//Arrange

int a=20;

int b=10;

//Act

double result=calculator.divide(a , b);

//Assert

assertEquals(2 , result);

}

@Test

void testIsPositive() {

assertTrue(calculator.isPositive(7));

assertFalse(calculator.isPositive(-1));

}

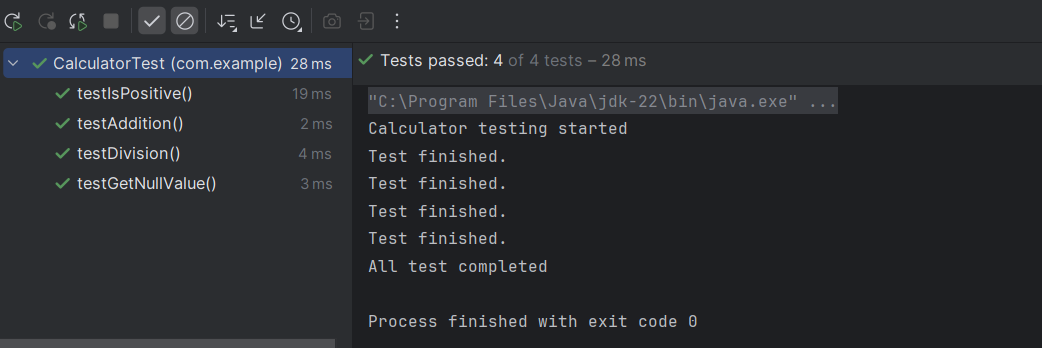
@Test

void testGetNullValue() {

assertNull(calculator.getNullValue());

}

}

OUTPUT:

Mokito Exercises :

Exercise 1: Mocking and Stubbing :

File name : WeatherService.java

(Acts as an external Api)

package com.example;

public interface WeatherService {

String getForecast(String city);

}

File Name: WeatherReporter.java

package com.example;

public class WeatherReporter {

private final WeatherService weatherService;

public WeatherReporter(WeatherService weatherService) {

this.weatherService = weatherService;

}

public String report(String city) {

String forecast = weatherService.getForecast(city);

return "Weather report for " + city + ": " + forecast;

}

}

File Name :WeatherReportTesting

package com.example;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class WeatherReportTesting {

private WeatherService mockWeatherService;

private WeatherReporter reporter;

@BeforeEach

void setUp() {

mockWeatherService = mock(WeatherService.class);

reporter = new WeatherReporter(mockWeatherService);

}

@Test

void testWeatherReport\_forChennai() {

// Stub the mock

when(mockWeatherService.getForecast("Chennai")).thenReturn("Sunny, 35°C");

// Act

String report = reporter.report("Chennai");

// Assert

assertEquals("Weather report for Chennai: Sunny, 35°C", report);

verify(mockWeatherService, times(1)).getForecast("Chennai");

}

@Test

void testWeatherReport\_forDelhi() {

when(mockWeatherService.getForecast("Delhi")).thenReturn("Cloudy, 28°C");

String report = reporter.report("Delhi");

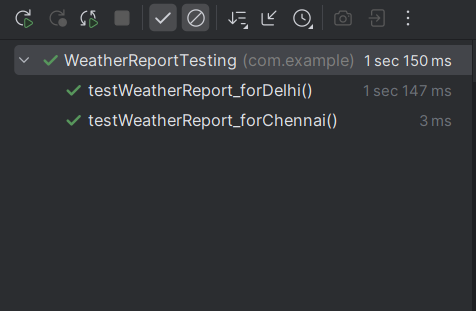
assertEquals("Weather report for Delhi: Cloudy, 28°C", report);

verify(mockWeatherService).getForecast("Delhi");

}

}

OUTPUT:



Exercise 2: Verifying Interactions:

File name: WeatherReporter.java

package com.example;

public class WeatherReporter {

private final WeatherService service;

public WeatherReporter(WeatherService service) {

this.service = service;

}

public String report(String city) {

return "Report: " + service.getForecast(city);

}

}

File name: WeatherReportTesting.java

package com.example;

import com.example.WeatherReporter;

import com.example.WeatherService;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class WeatherReportTesting {

private WeatherService mockService;

private WeatherReporter reporter;

@BeforeEach

void setUp() {

mockService = mock(WeatherService.class);

reporter = new WeatherReporter(mockService);

}

@Test

void testMockInteraction() {

// Arrange (Stubbing)

when(mockService.getForecast("Delhi")).thenReturn("Cloudy");

// Act

String result = reporter.report("Delhi");

// Assert return value

assertEquals("Report: Cloudy", result);

// Verify interaction with mock

verify(mockService).getForecast("Delhi");

// Verify it was called only once

verify(mockService, times(1)).getForecast("Delhi");

// Verify no other method was called

verifyNoMoreInteractions(mockService);

}

}

OUTPUT:

