JUnit Testing Exercises

# Exercise 1: Setting Up JUnit

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(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 IllegalArgumentException("Division by zero is not allowed.");

}

return a / b;

}

}

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

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

*assertEquals*(15, result);

}

@Test

public void testSubtract() {

Calculator calc = new Calculator();

int result = calc.subtract(10, 5);

*assertEquals*(5, result);

}

@Test

public void testMultiply() {

Calculator calc = new Calculator();

int result = calc.multiply(10, 5);

*assertEquals*(50, result);

}

@Test

public void testDivide() {

Calculator calc = new Calculator();

int result = calc.divide(10, 5);

*assertEquals*(2, result);

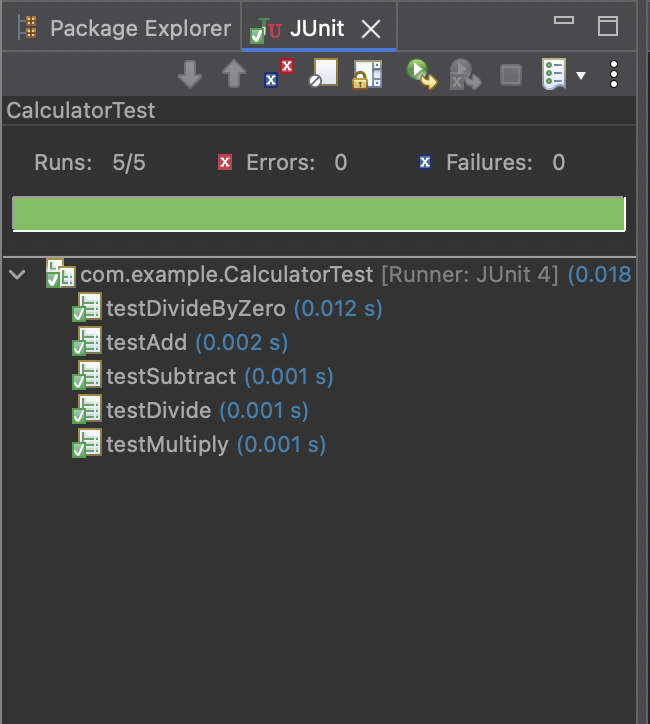
}

*@Test*(expected = IllegalArgumentException.class)

public void testDivideByZero() {

Calculator calc = new Calculator();

calc.divide(10, 0); // Should throw exception

}

}

# Exercise 3: Assertions in JUnit

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

// Assert equals

*assertEquals*("Sum should be 5", 5, 2 + 3);

// Assert true

*assertTrue*("5 should be greater than 3", 5 > 3);

// Assert false

*assertFalse*("5 should not be less than 3", 5 < 3);

// Assert null

Object obj1 = null;

*assertNull*("Object should be null", obj1);

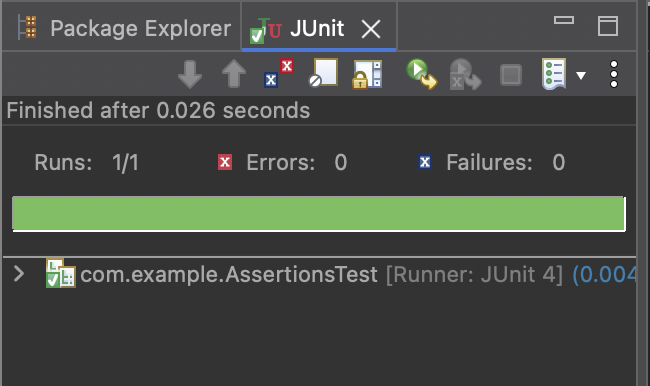
// Assert not null

Object obj2 = new Object();

*assertNotNull*("Object should not be null", obj2);

}

}



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

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(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 IllegalArgumentException("Division by zero is not allowed.");

}

return a / b;

}

}

package com.example;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorFixtureTest {

private Calculator calculator;

@Before

public void setUp() {

System.***out***.println("Setting up Calculator instance...");

calculator = new Calculator(); // Arrange

}

@After

public void tearDown() {

System.***out***.println("Tearing down Calculator instance...");

calculator = null;

}

@Test

public void testAdd() {

// Act

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

// Assert

*assertEquals*(15, result);

}

@Test

public void testSubtract() {

int result = calculator.subtract(10, 3);

*assertEquals*(7, result);

}

@Test

public void testMultiply() {

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

*assertEquals*(20, result);

}

@Test

public void testDivide() {

int result = calculator.divide(20, 4);

*assertEquals*(5, result);

}

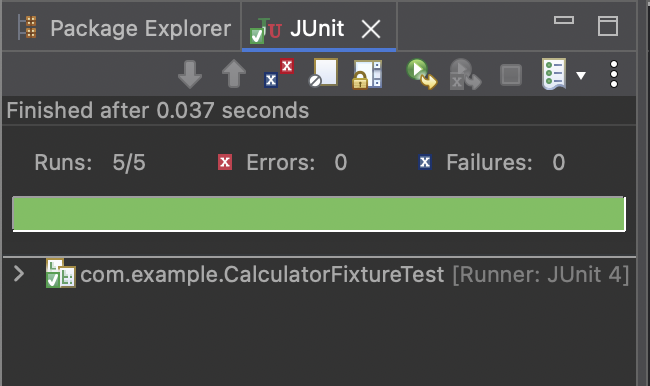
*@Test*(expected = IllegalArgumentException.class)

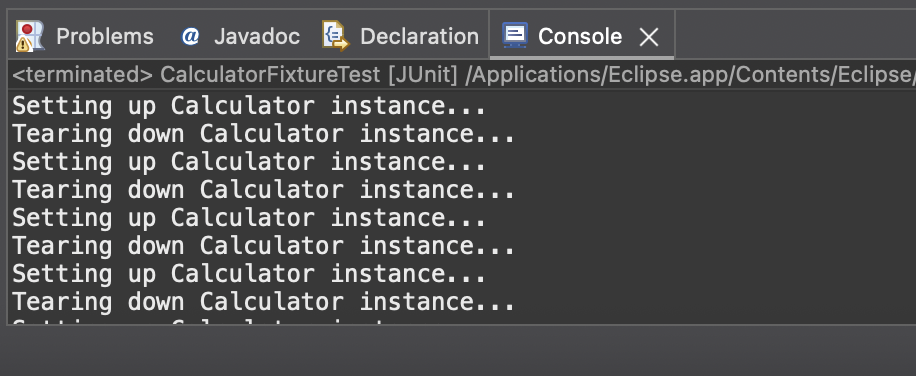
public void testDivideByZero() {

calculator.divide(10, 0); // Should throw exception

}

}





Mockito Hands-On Exercises

# Exercise 1: Mocking and Stubbing

package com.example;

public interface ExternalApi {

String getData();

}

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

package com.example;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest {

@Test

public void testExternalApi() {

// Create a mock of ExternalApi

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

// Stub getData() to return a mock value

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

// Inject mock into the service

MyService service = new MyService(mockApi);

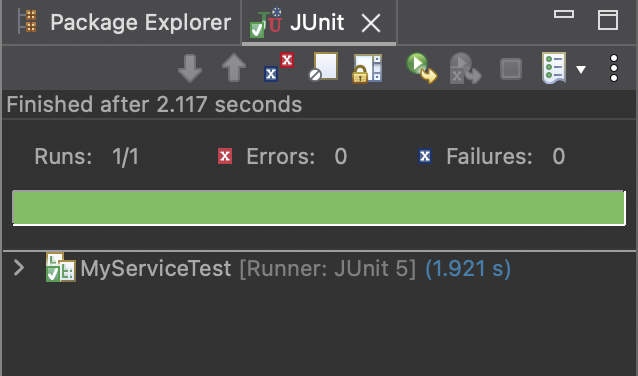
// Call method and assert

String result = service.fetchData();

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

}

}



# Exercise 2: Verifying Interactions

package com.example;

public interface ExternalApi {

String getData();

}

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Arrange: create a mock of ExternalApi

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

// Act: pass mock to service and call method

MyService service = new MyService(mockApi);

service.fetchData();

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

*verify*(mockApi).getData();

}

}

