**DN 4.0 JAVA FSE SOLUTIONS – WEEK 2**

**SKILL: TDD using JUnit5 and Mockito**

**JUnit\_Basic Testing Exercises**

**Exercise 1:** Setting Up JUnit

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

**Code:**

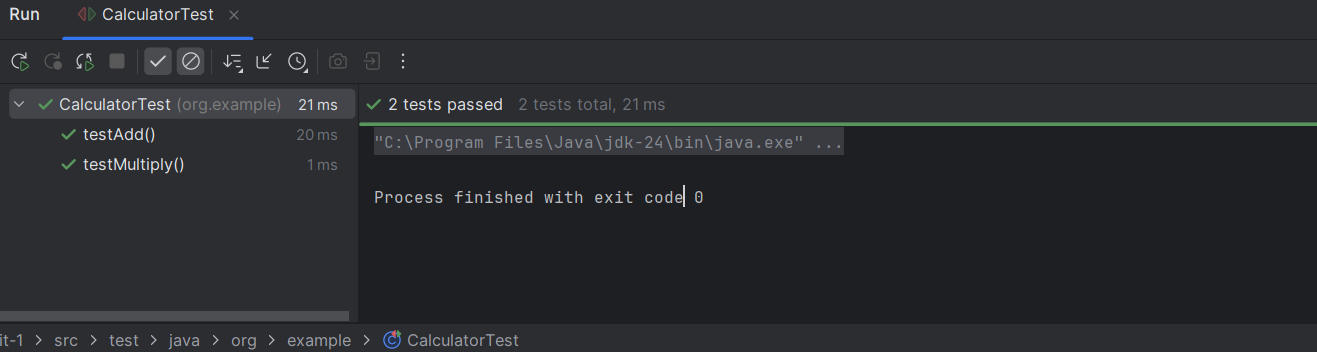
**Calculator.java**

package org.example;  
  
public class Calculator {  
 public int add(int a, int b) {  
 return a + b;  
 }  
  
 public int multiply(int a, int b) {  
 return a \* b;  
 }}

**CalculatorTest.java**

package org.example;  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
class CalculatorTest {  
  
 @Test  
 void testAdd() {  
 Calculator calc = new Calculator();  
 *assertEquals*(5, calc.add(2, 3), "2 + 3 should equal 5");  
 }  
  
 @Test  
 void testMultiply() {  
 Calculator calc = new Calculator();  
 *assertEquals*(12, calc.multiply(3, 4), "3 \* 4 should equal 12");  
 }}

**Output:**

****

**Exercise 3:** Assertions in JUnit

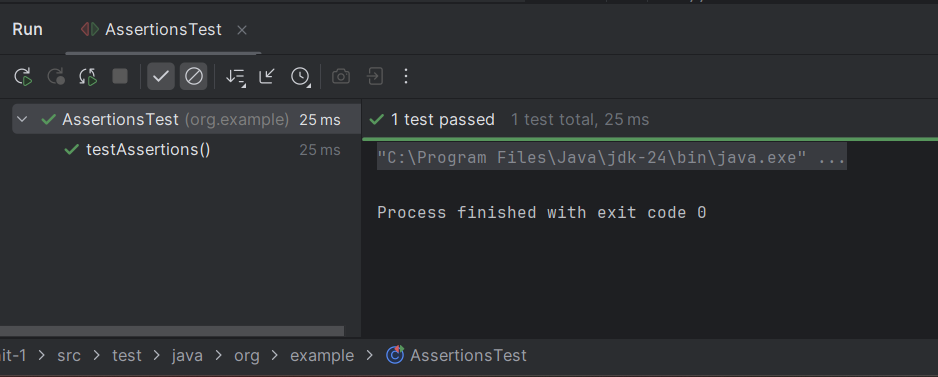
**Scenario**: You need to use different assertions in JUnit to validate your test results.

**Code:**

**AssertionsTest.java**

package org.example;  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
public class AssertionsTest {  
 @Test  
 public void testAssertions() {  
 // Assert equals  
 *assertEquals*(5, 2 + 3, "2 + 3 should be 5");  
 // Assert true  
 *assertTrue*(5 > 3, "5 is greater than 3");  
 // Assert false  
 *assertFalse*(5 < 3, "5 is not less than 3");  
 // Assert null  
 *assertNull*(null, "The object should be null");  
 // Assert not null  
 *assertNotNull*(new Object(), "The object should not be null");  
 }  
}

**Output:**



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

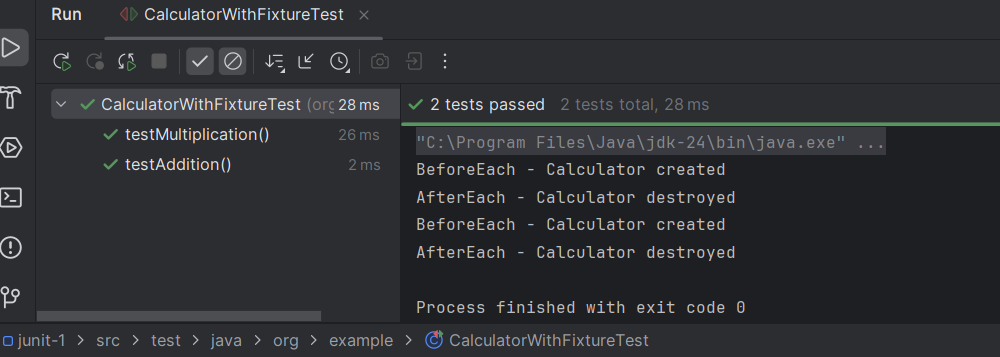
**Scenario:** You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

**Code:**

**CalculatorWithFixtureTest.java**

package org.example;  
import org.junit.jupiter.api.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
class CalculatorWithFixtureTest {  
 private Calculator calculator;  
 @BeforeEach  
 void setUp() {  
 // Arrange (common setup)  
 calculator = new Calculator();  
 System.*out*.println("BeforeEach - Calculator created");  
 }  
 @AfterEach  
 void tearDown() {  
 // Clean up after each test  
 calculator = null;  
 System.*out*.println("AfterEach - Calculator destroyed");  
 }  
 @Test  
 void testAddition() {  
 // Arrange done in setUp()  
 // Act  
 int result = calculator.add(10, 5);  
 // Assert  
 *assertEquals*(15, result, "10 + 5 should equal 15");  
 }  
 @Test  
 void testMultiplication() {  
 // Act  
 int result = calculator.multiply(4, 5);  
 // Assert  
 *assertEquals*(20, result, "4 \* 5 should equal 20");  
 }  
}

**Output:**



**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.

**Code:**

**ExternalApi.java**

package org.example;  
  
public interface ExternalApi {  
 String getData();  
}

**MyServices.java**

package org.example;  
  
public class MyService {  
 private final ExternalApi externalApi;  
  
 public MyService(ExternalApi externalApi) {  
 this.externalApi = externalApi;  
 }  
  
 public String fetchData() {  
 return externalApi.getData();  
 }  
}

**MyservicesTest.java**

package org.example;  
  
import org.junit.jupiter.api.Test;  
import org.mockito.Mockito;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.Mockito.*when*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testExternalApi() {  
 // 1. Create mock of ExternalApi  
 ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);  
  
 // 2. Stub method to return predefined value  
 when(mockApi.getData()).thenReturn("Mock Data");  
  
 // 3. Inject mock into MyService  
 MyService service = new MyService(mockApi);  
  
 // 4. Call method and assert result  
 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.

**Code:**

**ExternalApi.java**

package org.example;

public interface ExternalApi {

String getData();

}

**MyService.java**

package org.example;

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**MyServiceTest.java**

package org.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// 1. Create mock

ExternalApi mockApi = mock(ExternalApi.class);

// 2. Create service and call method

MyService service = new MyService(mockApi);

service.fetchData();

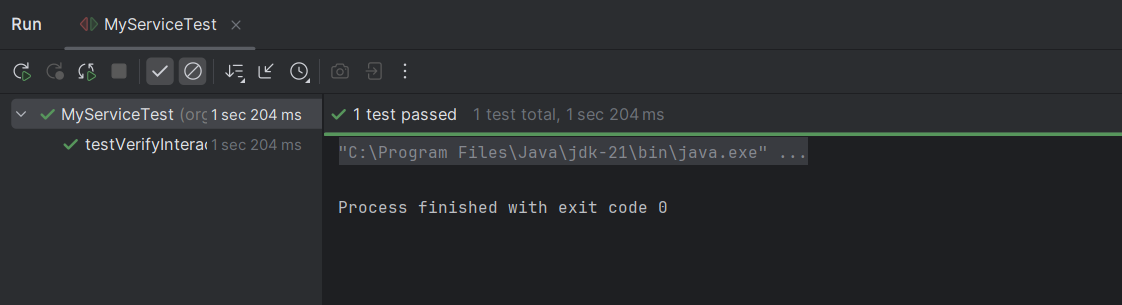
// 3. Verify that getData() was called

verify(mockApi).getData();

}

}

**Output:**

****