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

**SKILL: TDD using JUnit5 and Mockito**

**JUnit Testing Exercises:**

**Exercise 1:** Setting Up Junit

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

**Code:**

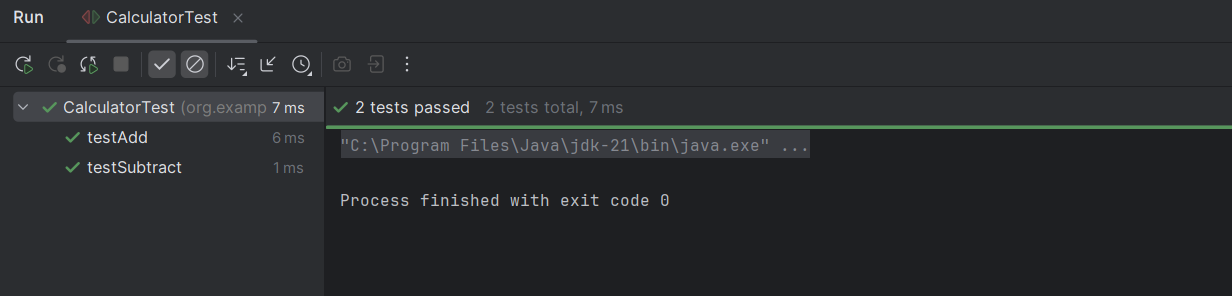
**Calculator.java –**

package org.example;  
  
public class Calculator {  
 public int add(int a, int b) {  
 return a + b;  
 }  
  
 public int subtract(int a, int b) {  
 return a - b;  
 }  
}

**CalculatorTest.java -**

package org.example;  
import org.junit.Test;  
import static org.junit.Assert.*assertEquals*;  
  
public class CalculatorTest {  
  
 @Test  
 public void testAdd() {  
 Calculator calc = new Calculator();  
 *assertEquals*(5, calc.add(2, 3));  
 }  
  
 @Test  
 public void testSubtract() {  
 Calculator calc = new Calculator();  
 *assertEquals*(1, calc.subtract(4, 3));  
 }  
}

**Output:**

****

**Exercise 3:** Assertions in Junit

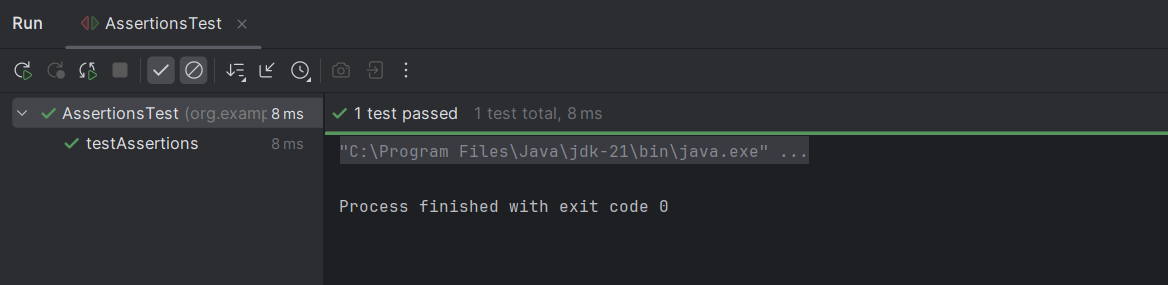
**Scenario:** You need to use different assertions in JUnit to validate your test results.

**Code:**

**AssertionsTest.java –**

package org.example;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
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());  
 }  
}

**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:**

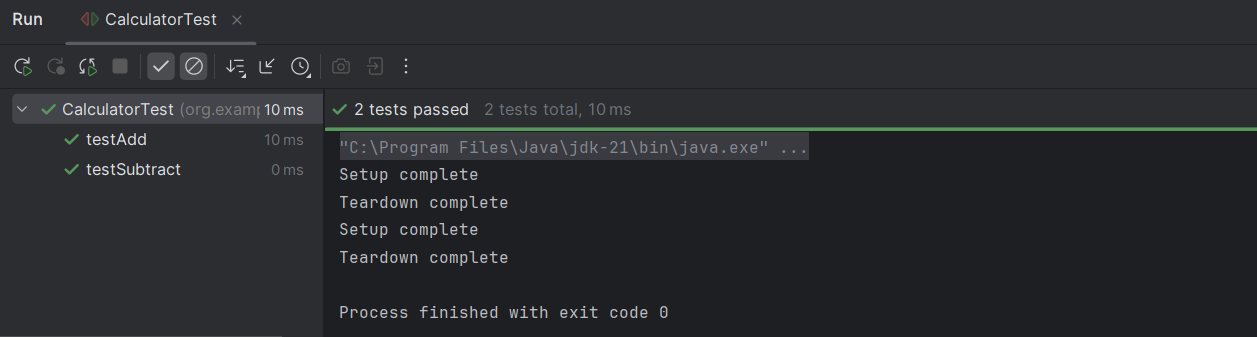
**Calculator.java –**

package org.example;  
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 org.example;  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
public class CalculatorTest {  
 private Calculator calculator;  
 // Setup method – runs before every @Test  
 @Before  
 public void setUp() {  
 calculator = new Calculator(); // Arrange: setup the object  
 System.*out*.println("Setup complete");  
 }  
 // Teardown method – runs after every @Test  
 @After  
 public void tearDown() {  
 calculator = null;  
 System.*out*.println("Teardown complete");  
 }  
  
 @Test  
 public void testAdd() {  
 // Act  
 int result = calculator.add(2, 3);  
  
 // Assert  
 *assertEquals*(5, result);  
 }  
  
 @Test  
 public void testSubtract() {  
 // Act  
 int result = calculator.subtract(5, 3);  
 // Assert  
 *assertEquals*(2, result);  
 }  
}

**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.

**Code:**

**ExternalApi.java -**

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

**MyService.java -**

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

**MyServiceTest.java -**

package org.example;  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.Mockito.\*;  
import org.junit.jupiter.api.Test;  
import org.mockito.Mockito;  
public class MyServiceTest {  
 @Test  
 public void testExternalApi() {  
 // Step 1: Create mock  
 ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);  
 // Step 2: Stub method  
 *when*(mockApi.getData()).thenReturn("Mock Data");  
 // Step 3: Inject mock and test  
 MyService service = new MyService(mockApi);  
 String result = service.fetchData();  
 // Assertion  
 *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();  
}

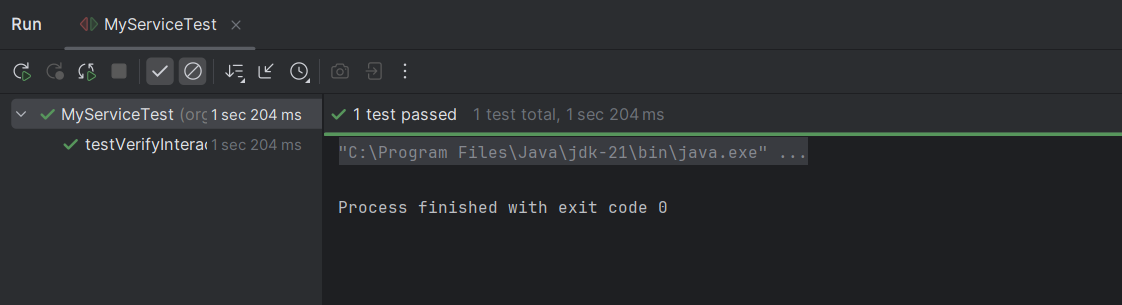
**MyService.java -**

package org.example;  
public class MyService {  
 private final ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
 public String fetchData() {  
 return api.getData(); // This method should trigger the interaction  
 }  
}

**MyServiceTest.java –**

package org.example;  
import static org.mockito.Mockito.\*;  
import org.junit.jupiter.api.Test;  
import org.mockito.Mockito;  
public class MyServiceTest {  
 @Test  
 public void testVerifyInteraction() {  
 // Step 1: Create mock  
 ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);  
 // Step 2: Use the service and call the method  
 MyService service = new MyService(mockApi);  
 service.fetchData();  
 // Step 3: Verify interaction  
 *verify*(mockApi).getData(); // verifies getData() was called exactly once  
 }  
}

**Output:**

****