**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:**

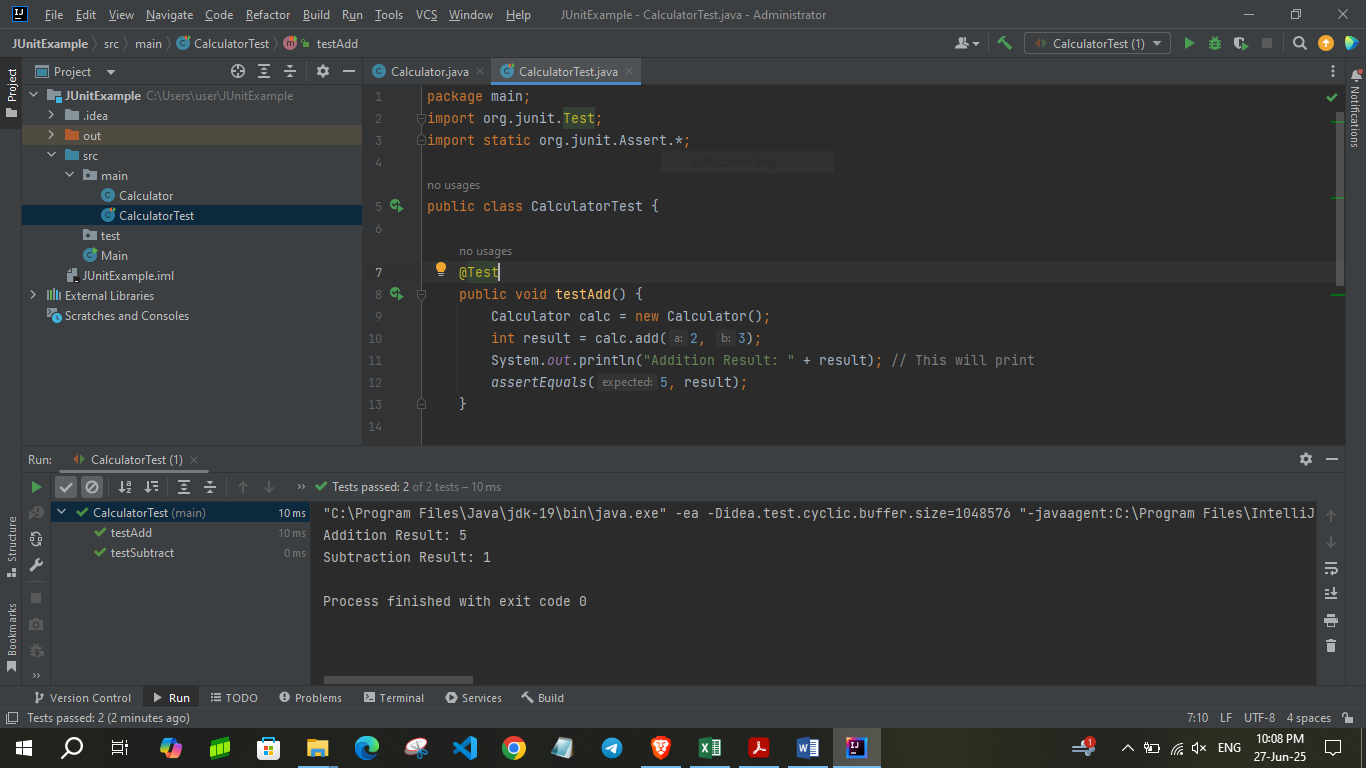
**Calculator.java (main class):**

package main;  
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 (test class):**

package main;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class CalculatorTest {  
  
 @Test  
 public void testAdd() {  
 Calculator calc = new Calculator();  
 int result = calc.add(2, 3);  
 System.*out*.println("Addition Result: " + result); // This will print  
 *assertEquals*(5, result);  
 }  
  
 @Test  
 public void testSubtract() {  
 Calculator calc = new Calculator();  
 int result = calc.subtract(4, 3);  
 System.*out*.println("Subtraction Result: " + result); // This will print  
 *assertEquals*(1, result);  
 }  
}

**OUTPUT:**



Exercise 3: Assertions in Junit

Scenario:

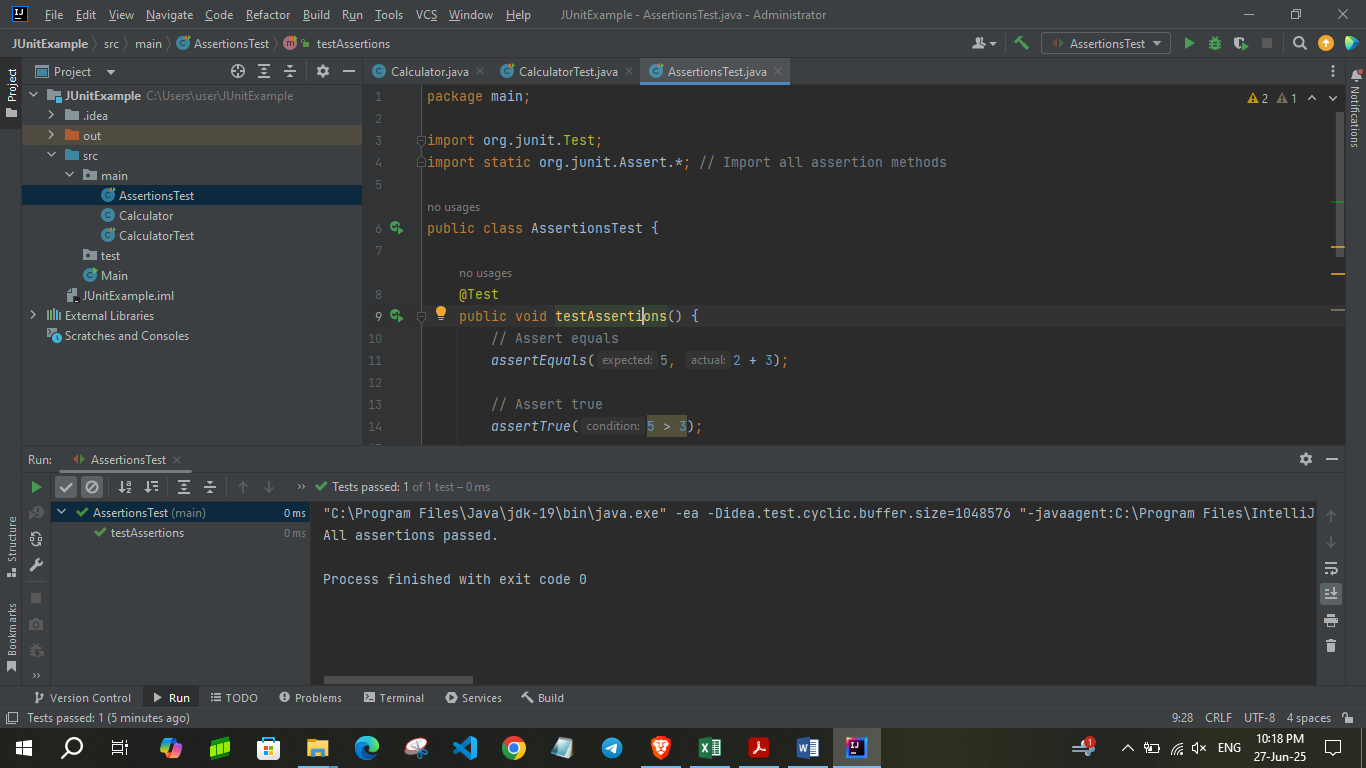
You need to use different assertions in JUnit to validate your test results.

**CODE:**

**AssertionsTest.java:**

package main;  
  
import org.junit.Test;  
import static org.junit.Assert.\*; // Import all assertion methods  
  
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  
 Object obj1 = null;  
 *assertNull*(obj1);  
  
 // Assert not null  
 Object obj2 = new Object();  
 *assertNotNull*(obj2);  
  
 // Optional: print something to verify visually (for learning)  
 System.*out*.println("All assertions passed.");  
 }  
}

**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:**

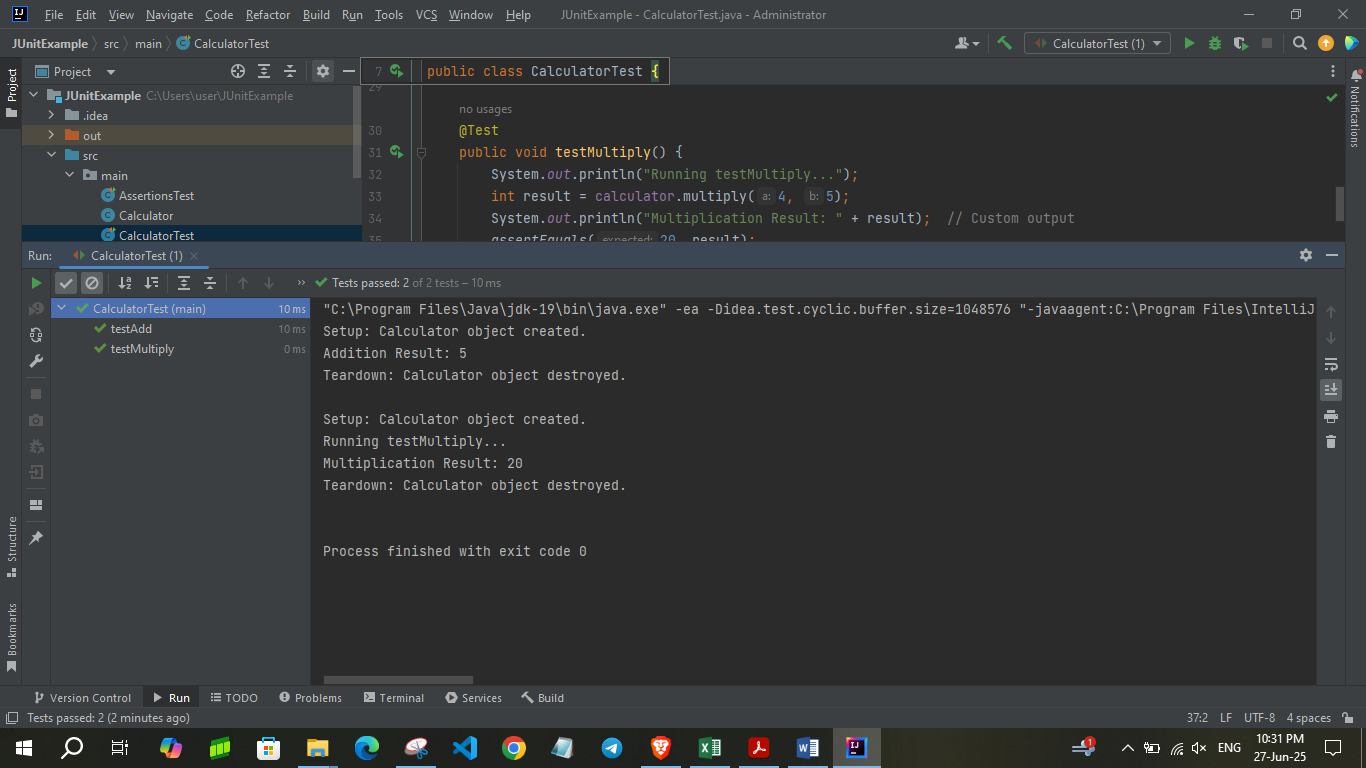
**Calculator.java:**

package main;  
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 main;  
import org.junit.Before;  
import org.junit.After;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class CalculatorTest {  
  
 private Calculator calculator;  
  
 @Before  
 public void setUp() {  
 calculator = new Calculator();  
 System.*out*.println("Setup: Calculator object created.");  
 }  
  
 @After  
 public void tearDown() {  
 calculator = null;  
 System.*out*.println("Teardown: Calculator object destroyed.\n");  
 }  
  
 @Test  
 public void testAdd() {  
 int result = calculator.add(2, 3);  
 System.*out*.println("Addition Result: " + result); // Custom output  
 *assertEquals*(5, result);  
 }  
  
 @Test  
 public void testMultiply() {  
 System.*out*.println("Running testMultiply...");  
 int result = calculator.multiply(4, 5);  
 System.*out*.println("Multiplication Result: " + result); // Custom output  
 *assertEquals*(20, result);  
 }  
}

**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 main;  
  
public interface ExternalApi {  
 String getData();  
}

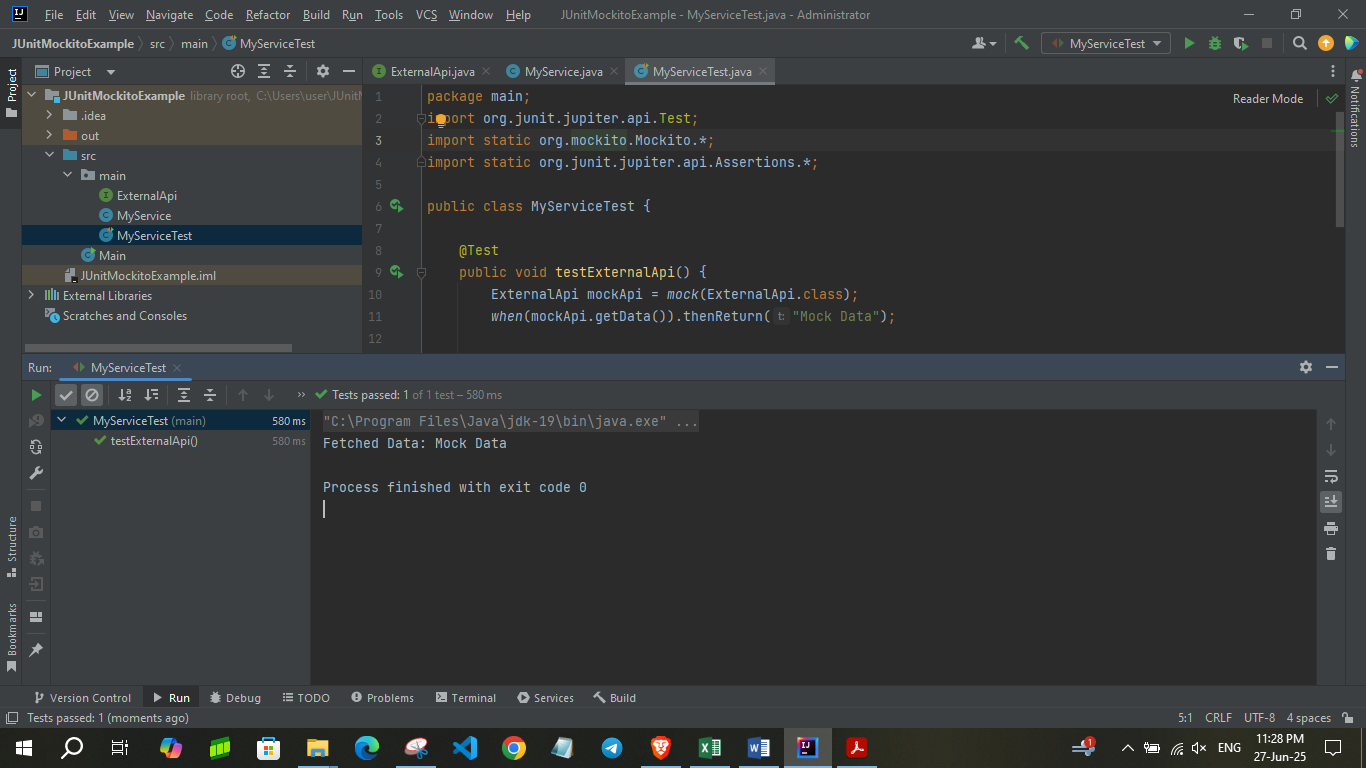
**MyService.java**

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

**MyServiceTest.java**

package main;  
  
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();  
 System.*out*.println("Fetched Data: " + result);  
 *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**

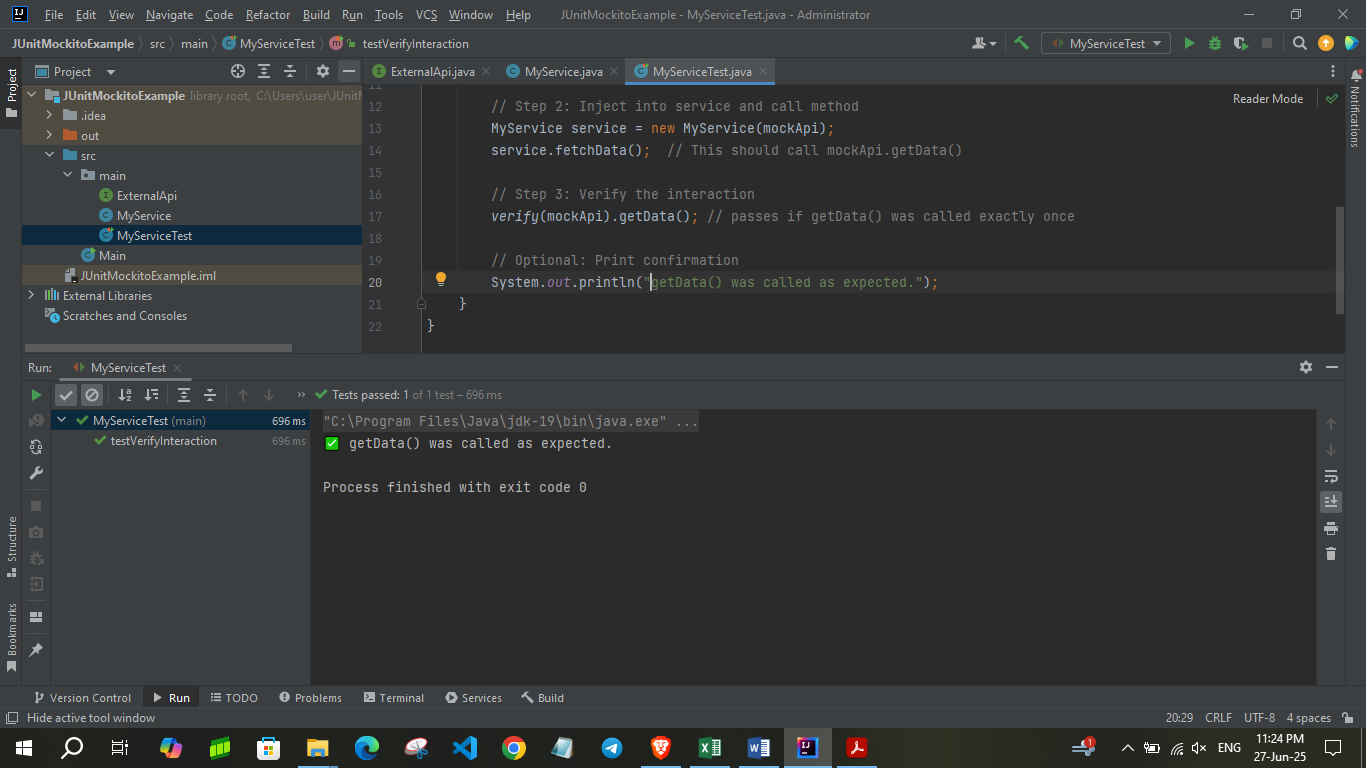
package main;  
  
public interface ExternalApi {  
 String getData();  
}

**MyService.java**

package main;  
public class MyService {  
 private ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData(); // calls the mock  
 }**MyServiceTest.java**

package main;  
import org.junit.Test;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testVerifyInteraction() {  
 // Step 1: Create a mock  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 // Step 2: Inject into service and call method  
 MyService service = new MyService(mockApi);  
 service.fetchData(); // This should call mockApi.getData()  
  
 // Step 3: Verify the interaction  
 *verify*(mockApi).getData(); // passes if getData() was called exactly once  
  
 // Optional: Print confirmation  
 System.*out*.println("✅ getData() was called as expected.");  
 }  
}

**OUTPUT:**



**SL4J Logging exercises**

Exercise 1: Logging Error Messages and Warning Levels

**Task:** Write a Java application that demonstrates logging error messages and warning levels

using SLF4J.

**CODE:**

**Logging.java**

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
public class Logging {  
 private static final Logger *logger* = LoggerFactory.*getLogger*(Logging.class);  
  
 public static void main(String[] args) {  
 *logger*.error("This is an error message");  
 *logger*.warn("This is a warning message");  
 }  
}

**OUTPUT:**

