**TDD using JUnit5 and Mockito  
  
Exercise 1: Setting Up Junit  
  
Code:  
  
Calculator.java:**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:**import static org.junit.Assert.\*;

import org.junit.Test;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

assertEquals(5, calc.add(2, 3));

}

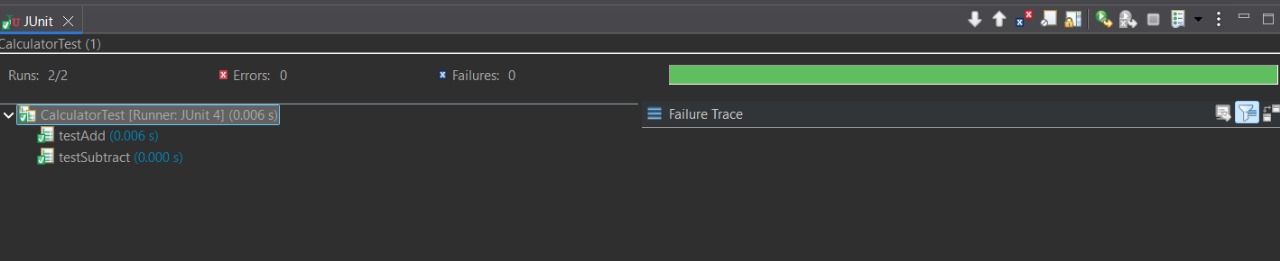
@Test

public void testSubtract() {

Calculator calc = new Calculator();

assertEquals(1, calc.subtract(5, 4));

}

}  
  
**OUTPUT:**

**Exercise 3: Assertions in Junit  
  
Code:**

**AssertionsTest.java:**import static org.junit.Assert.\*;

import org.junit.Test;

public class AssertionsTest {

@Test

public void testAssertions() {

// Assert equals

assertEquals(5, 2 + 3);

// Assert true

assertTrue(5 > 3);

// Assert false

assertFalse(5 < 3);

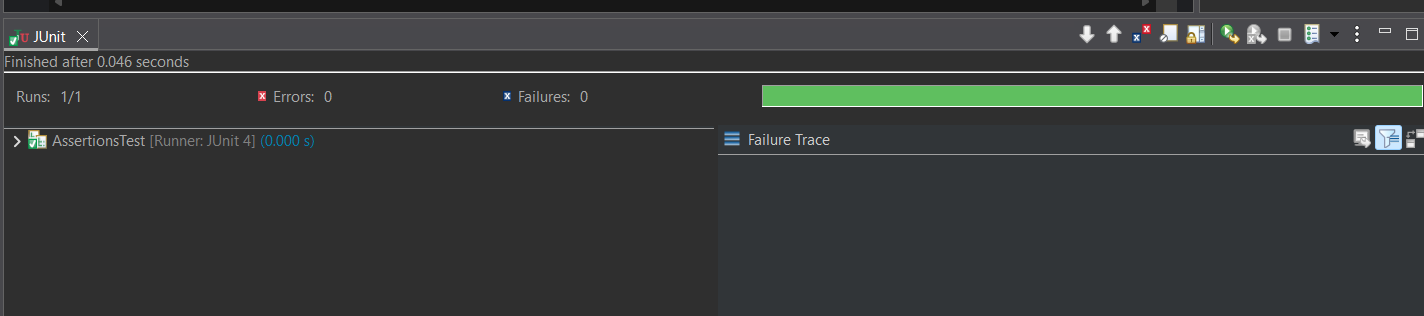
// Assert null

assertNull(null);

// Assert not null

assertNotNull(new Object());

}

}  
**Output:**

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

**Code:**

**Calculator.java:**

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:**

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.*assertEquals*;

public class CalculatorTest {

private Calculator calculator;

// 🔹 Setup before each test

*@Before*

public void setUp() {

calculator = new Calculator();

System.***out***.println("Setup done.");

}

// 🔹 Teardown after each test

*@After*

public void tearDown() {

System.***out***.println("Cleanup done.");

}

*@Test*

public void testAddition() {

// Arrange

int a = 5;

int b = 3;

// Act

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

// Assert

*assertEquals*(8, result);

}

*@Test*

public void testSubtraction() {

// Arrange

int a = 10;

int b = 4;

// Act

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

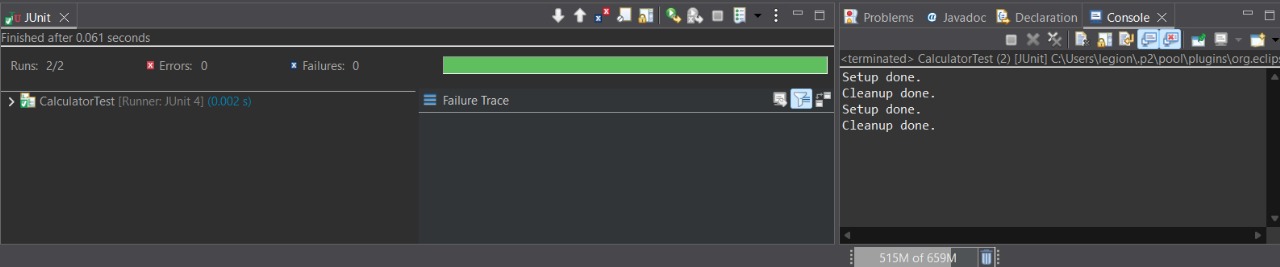
// Assert

*assertEquals*(6, result);

}

}

**Output:**



**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing  
Code :**

**ExternalApi.java:**

package com.example.junit\_test\_project2;

public interface ExternalApi {

String getData();

}

**MyService.java:**

package com.example.junit\_test\_project2;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**MyServiceTest.java:**

package com.example.junit\_test\_project2;

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

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testExternalApi() {

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

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

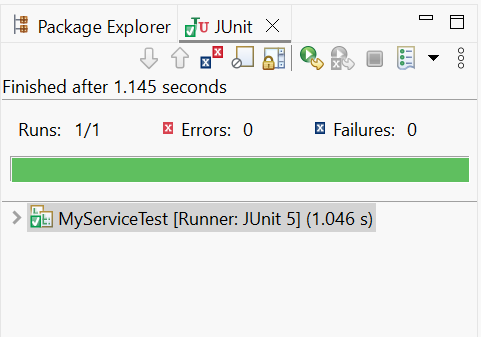
MyService service = new MyService(mockApi);

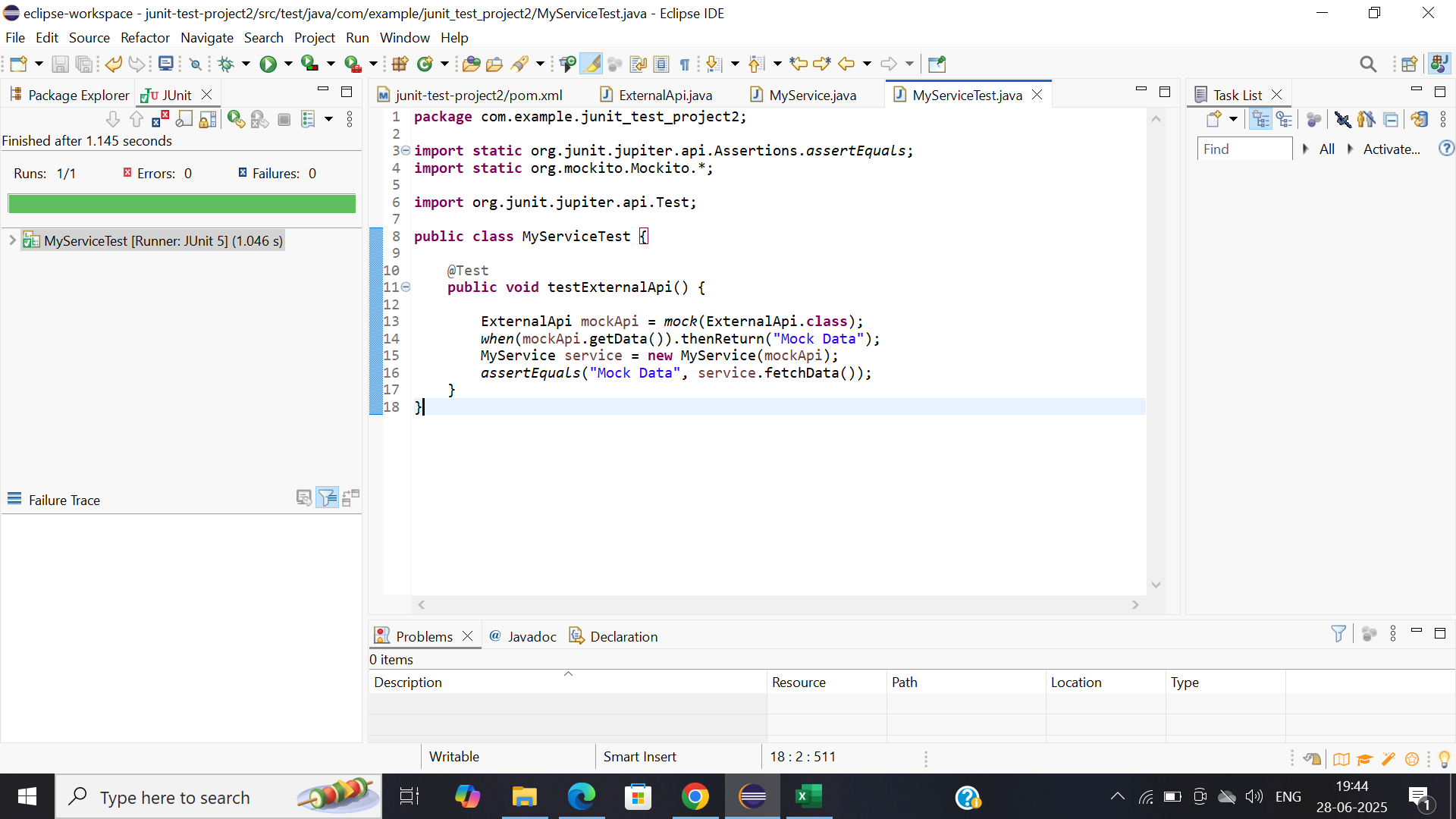
*assertEquals*("Mock Data", service.fetchData());

}

}

**OUTPUT:**

****

****

**Exercise 2: Verifying Interactions**

**Scenario:**

**You need to ensure that a method is called with specific arguments.**

**UserApi.java:**

package com.example.junit\_test\_project2;

public interface UserApi {

String getInfo();

**String getUserInfo(String userId);**

}

**UserService.java:**

package com.example.junit\_test\_project2;

public class UserService {

private UserApi userApi;

public UserService(UserApi userApi) {

this.userApi = userApi;

}

public String fetchInfo() {

return userApi.getInfo();

}

public String fetchUser(String userId) {

return userApi.getUserInfo(userId);

}

}

**UserServiceTest.java:**

package com.example.junit\_test\_project2;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class UserServiceTest {

@Test

public void testVerifySimpleInteraction() {

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

UserService service = new UserService(mockApi);

service.fetchInfo();

*verify*(mockApi).getInfo();

}

@Test

public void testVerifyMethodWithArgument() {

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

UserService service = new UserService(mockApi);

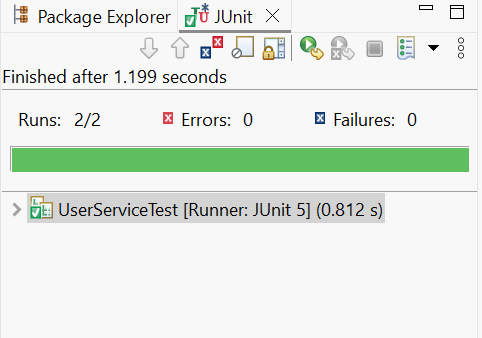
service.fetchUser("abc123");

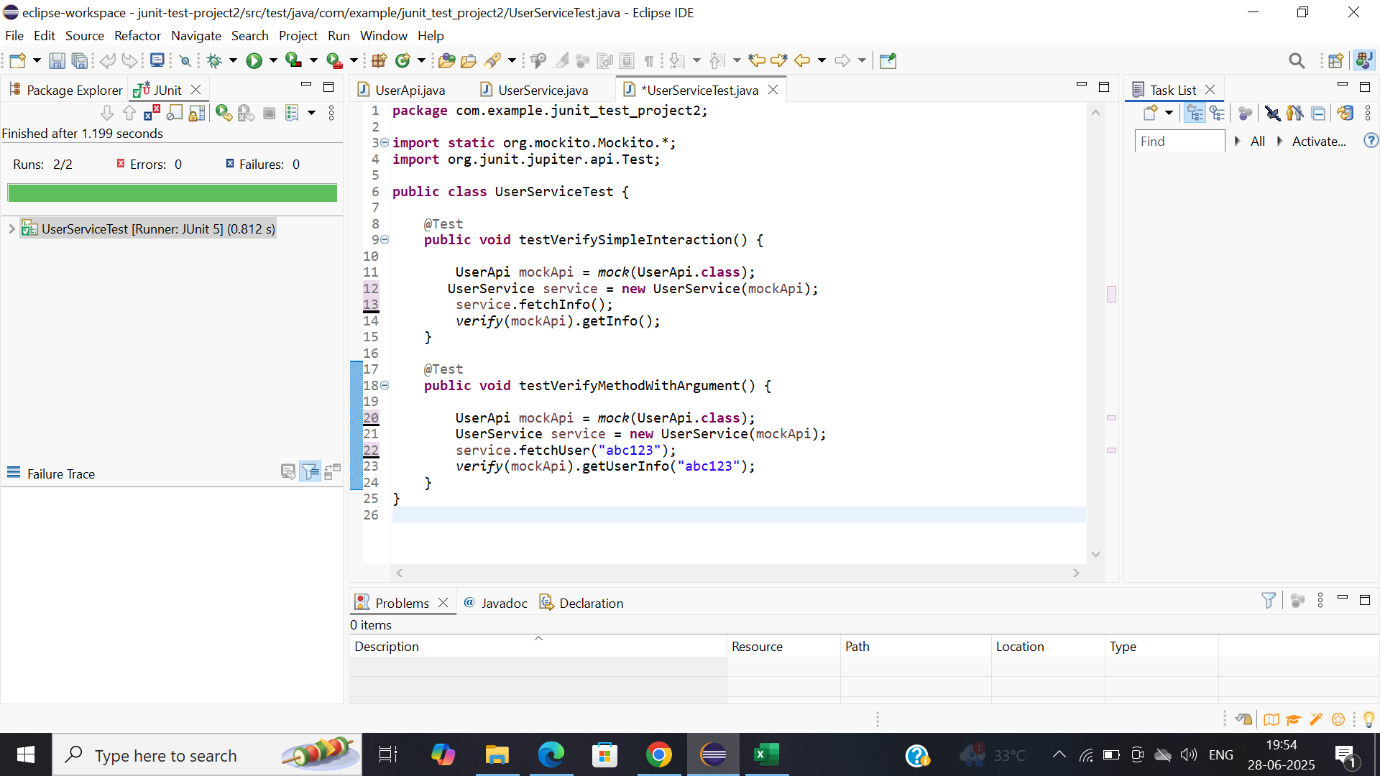
*verify*(mockApi).getUserInfo("abc123");

}

}

**OUTPUT:**

****

****

**v**