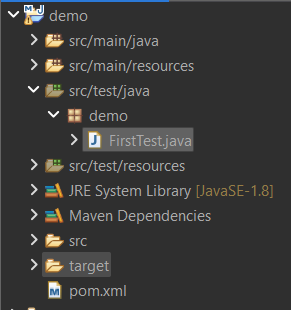
**JUNIT TESTING EXERCISES**

**Exercise 1: (Setting Up Junit) (Mandatory)**

**Folder Structure:**



**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.sample</groupId>

<artifactId>demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**FirstTest.java:**

package demo;

import org.junit.Test;

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

public class FirstTest {

*@Test*

public void additionTest() {

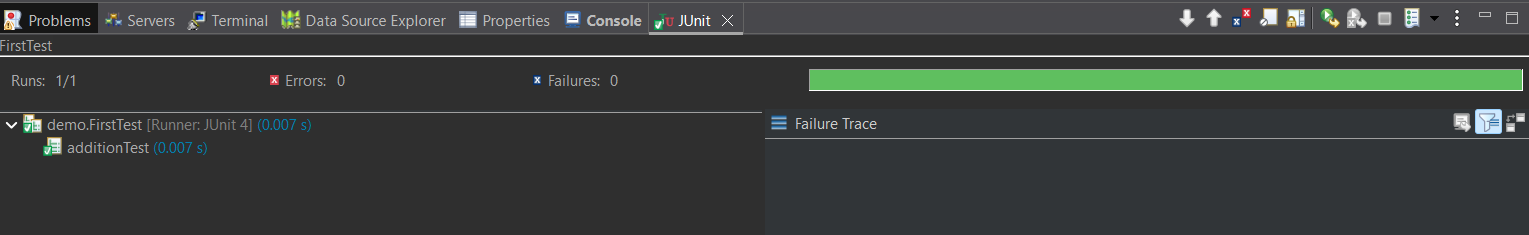
int result = 2 + 2;

*assertEquals*(4, result);

}

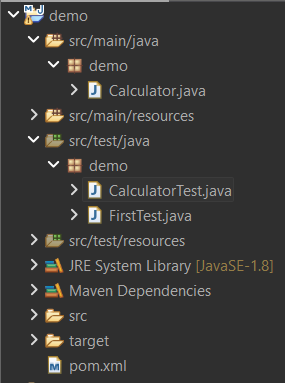
}

**Output:**

****

**Exercise 2: (Writing Basic Junit Tests)**

**Folder Structure:**

****

**Calculator.java:**

package demo;

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("Cannot divide by zero");

}

return a / b;

}

}

**CalculatorTest.java:**

package demo;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

Calculator calc = new Calculator();

*@Test*

public void testAddition() {

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

}

*@Test*

public void testSubtraction() {

*assertEquals*(2, calc.subtract(5, 3));

}

*@Test*

public void testMultiplication() {

*assertEquals*(15, calc.multiply(3, 5));

}

*@Test*

public void testDivision() {

*assertEquals*(2, calc.divide(10, 5));

}

*@Test*(expected = IllegalArgumentException.class)

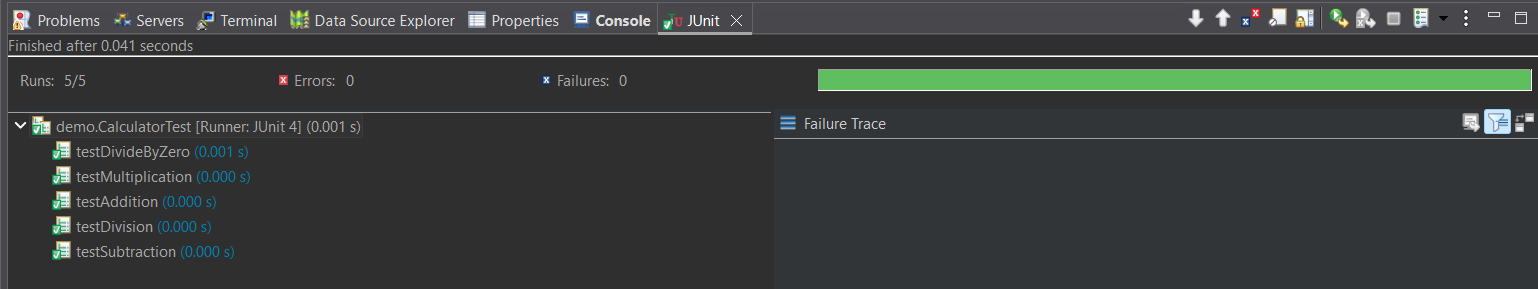
public void testDivideByZero() {

calc.divide(10, 0);

}

}

**Output:**



**Exercise 3: (Assertions in Junit)(Mandatory)**

**AssertionsTest.java:**

package demo;

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

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

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

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

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

import org.junit.Test;

public class AssertionsTest {

*@Test*

public void testAssertions() {

*assertEquals*(5, 2 + 3);

*assertTrue*(5 > 3);

*assertFalse*(5 < 3);

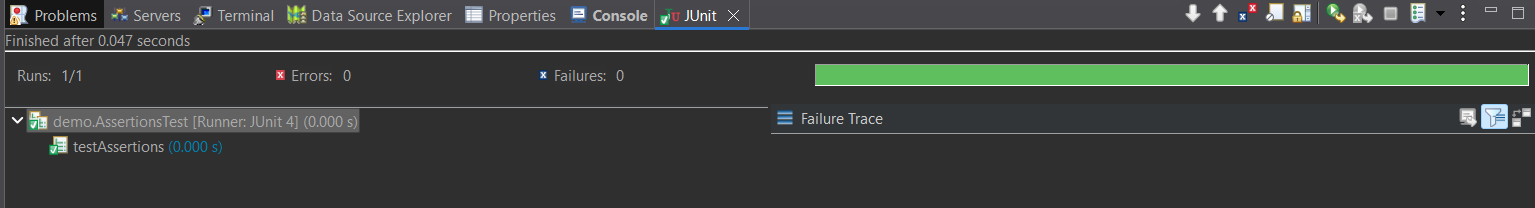
*assertNull*(null);

*assertNotNull*(new Object());

}

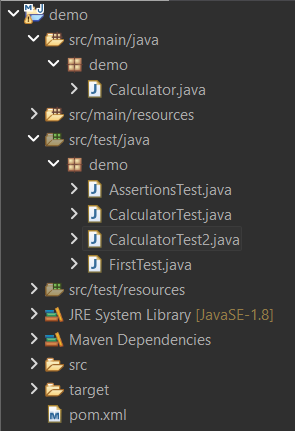
}

**Output:**



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

**Folder Structure:**

****

**CalculatorTest2.java:**

package demo;

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

import org.junit.Test;

import org.junit.After;

import org.junit.Before;

public class CalculatorTest2 {

private Calculator calculator;

*@Before*

public void setUp() {

calculator = new Calculator();

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

}

*@After*

public void tearDown() {

calculator = null;

System.***out***.println("Teardown complete");

}

*@Test*

public void testAddition() {

int a = 2;

int b = 3;

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

*assertEquals*(5, result);

}

*@Test*

public void testSubtraction() {

int a = 5;

int b = 3;

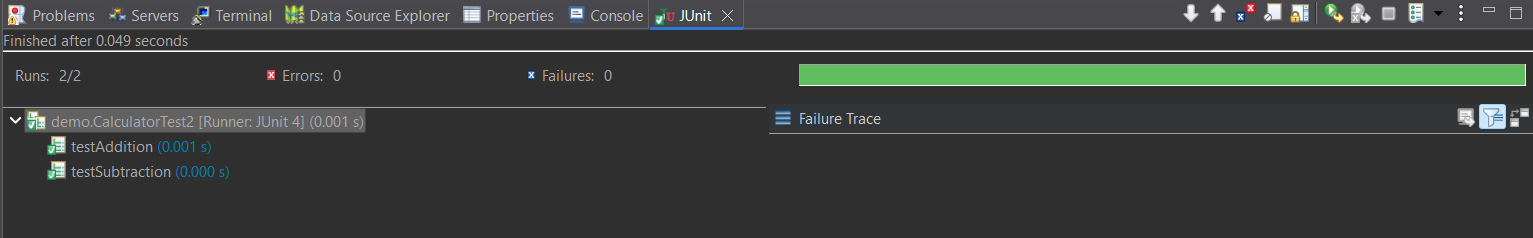
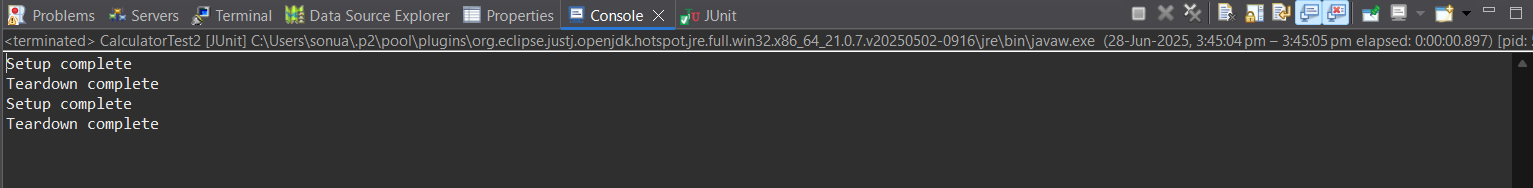
int result = calculator.subtract(a, b);

*assertEquals*(2, result);

}

}

**Output:**

**ADVANCED JUNIT TESTING EXERCISES**

**Exercise 1: (Parameterized Tests)**

**EvenChecker.java:**

package demo2;

public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**EvenCheckerTest.java:**

package demo2;

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

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.ValueSource;

public class EvenCheckerTest {

EvenChecker checker = new EvenChecker();

*@ParameterizedTest*

*@ValueSource*(ints = {2, 4, 6, 8, 10, 12})

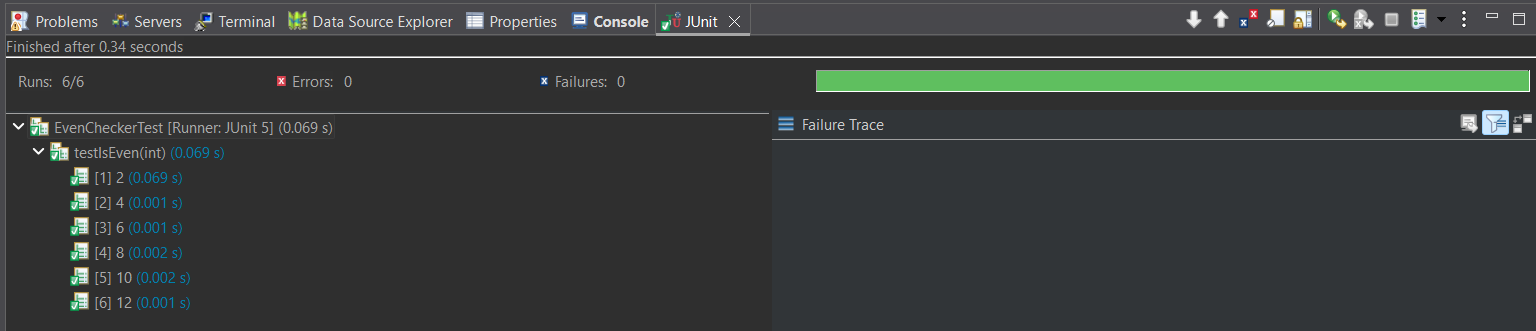
void testIsEven(int number) {

*assertTrue*(checker.isEven(number), number + " should be even");

}

}

**Output:**

****

**Exercise 2: (Test suites and Categories)**

**AllTests.java: (Test Suite)**

package demo;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

*@RunWith*(Suite.class)

*@*Suite.*SuiteClasses*({

FirstTest.class,

CalculatorTest.class,

CalculatorTest2.class,

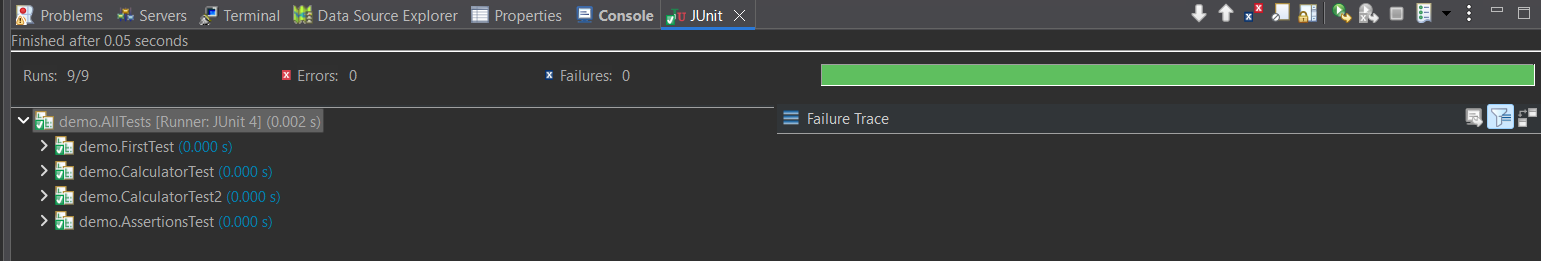
AssertionsTest.class

})

public class AllTests {

}

**Output:**

****

**Exercise 3: (Test Execution Order)**

**OrderedTests.java:**

package demo2;

import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;

import org.junit.jupiter.api.Order;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestMethodOrder;

*@TestMethodOrder*(OrderAnnotation.class)

public class OrderedTests {

*@Test*

*@Order*(1)

void testLogin() {

System.***out***.println("Running testLogin");

}

*@Test*

*@Order*(2)

void testAddItem() {

System.***out***.println("Running testAddItem");

}

*@Test*

*@Order*(3)

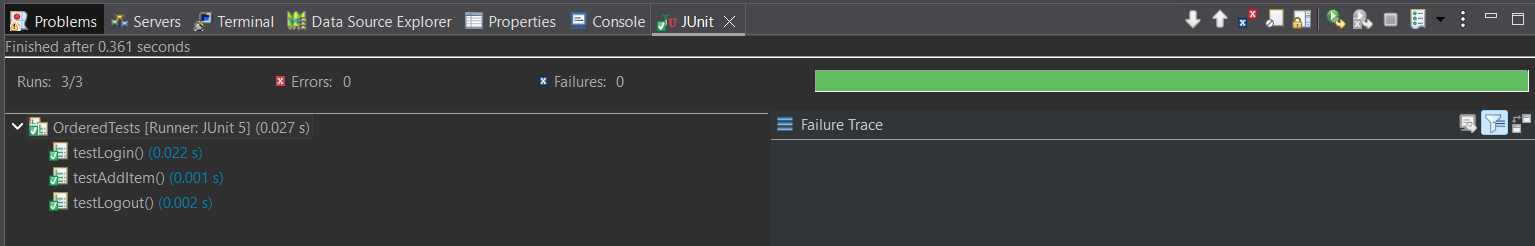
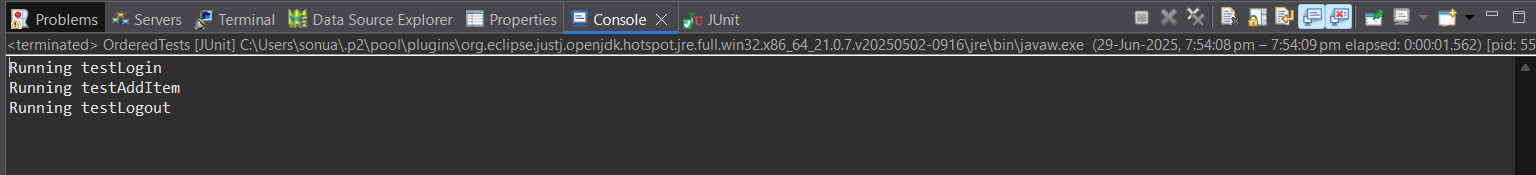
void testLogout() {

System.***out***.println("Running testLogout");

}

}

**Output:**

**** ****

**Exercise 4: (Exception Testing)**

**ExceptionThrower.java:**

package demo2;

public class ExceptionThrower {

public void throwException() {

throw new IllegalArgumentException("Invalid input provided");

}

}

**ExceptionThrowerTest.java:**

package demo2;

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

import org.junit.jupiter.api.Test;

public class ExceptionThrowerTest {

*@Test*

void testThrowsIllegalArgumentException() {

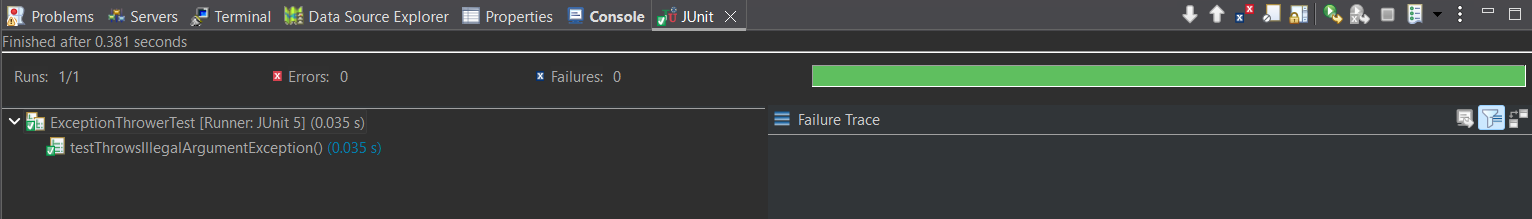
ExceptionThrower thrower = new ExceptionThrower();

*assertThrows*(IllegalArgumentException.class, thrower::throwException);

}

}

**Output:**

****

**Exercise 5: (Timeout and Performance Testing)**

**PerformanceTester.java:**

package demo2;

public class PerformanceTester {

public void performTask() throws InterruptedException {

System.***out***.println("Task started...");

Thread.*sleep*(400);

System.***out***.println("Task completed.");

}

}

**PerformanceTesterTest.java:**

package demo2;

import java.time.Duration;

import org.junit.jupiter.api.Test;

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

public class PerformanceTesterTest {

*@Test*

void testPerformTaskCompletesInTime() {

PerformanceTester tester = new PerformanceTester();

// Timeout test: will fail if performTask() takes more than 500ms

*assertTimeout*(Duration.*ofMillis*(500), () -> {

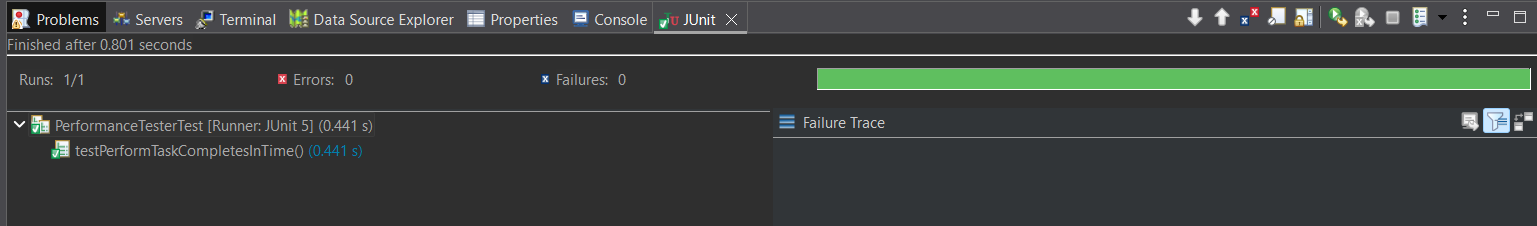
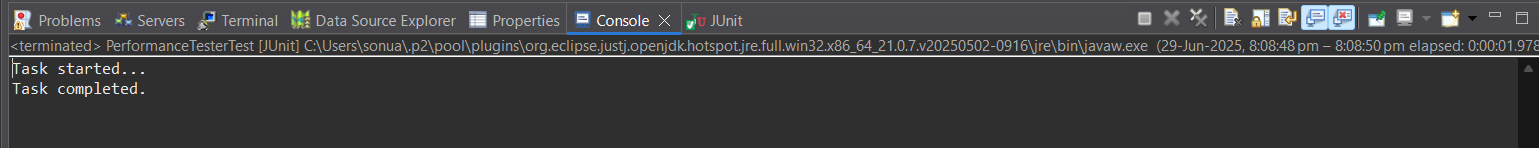
tester.performTask();

});

}

}

**Output:**

**** ****

**MOCKITO HANDS-ON EXERCISES**

**Exercise 1: (Mocking and Stubbing) (Mandatory)**

**ExternalApi.java: (Interface)**

package demo2;

public interface ExternalApi {

String getData();

}

**MyService.java:**

package demo2;

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**MyServiceTest.java:**

package demo2;

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

import static org.mockito.Mockito.*mock*;

import static org.mockito.Mockito.*when*;

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);

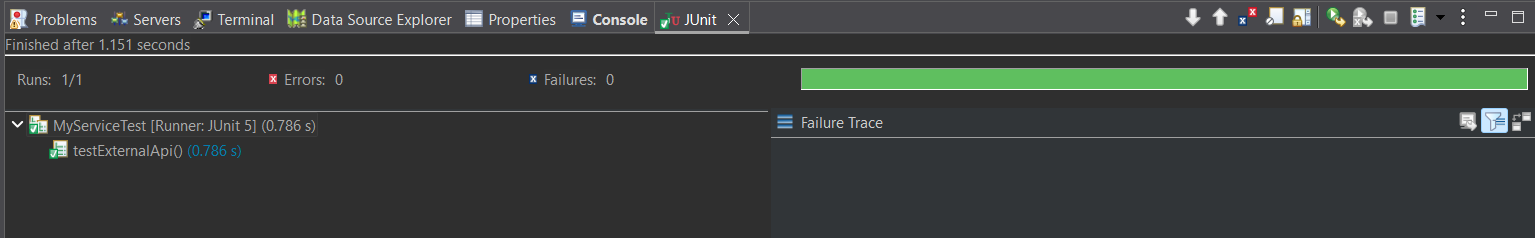
String result = service.fetchData();

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

}

}

**Output:**

****

**Exercise 2: (Verify Interactions) (Mandatory)**

**MyServiceTest.java:**

package demo2;

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

import static org.mockito.Mockito.*mock*;

import static org.mockito.Mockito.*verify*;

import static org.mockito.Mockito.*when*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

*@Test*

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApi);

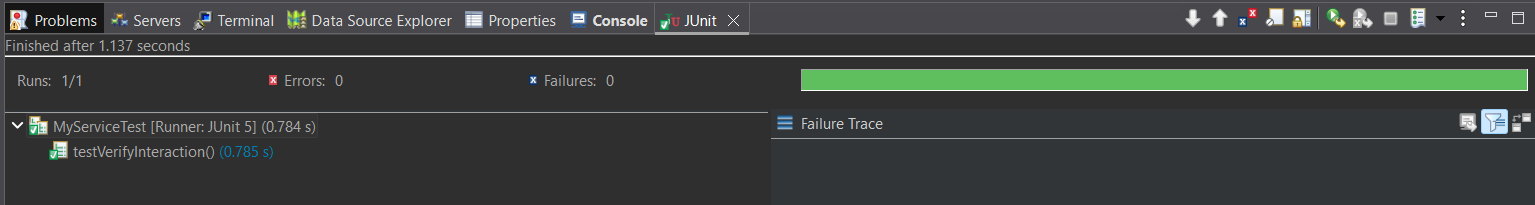
service.fetchData();

*verify*(mockApi).getData();

}

}

**Output:**

****

**Exercise 3: (Argument Matching)**

**NotificationService.java: (Interface)**

package demo2;

public interface NotificationService {

void send(String email, String message);

}

**UserService.java:**

package demo2;

public class UserService {

private NotificationService notificationService;

public UserService(NotificationService notificationService) {

this.notificationService = notificationService;

}

public void registerUser(String email) {

notificationService.send(email, "Welcome to our platform!");

}

}

**UserServiceTest.java:**

package demo2;

import static org.mockito.ArgumentMatchers.*eq*;

import static org.mockito.Mockito.*mock*;

import static org.mockito.Mockito.*verify*;

import org.junit.jupiter.api.Test;

public class UserServiceTest {

*@Test*

public void testSendCalledWithCorrectArguments() {

NotificationService mockNotification = *mock*(NotificationService.class);

UserService service = new UserService(mockNotification);

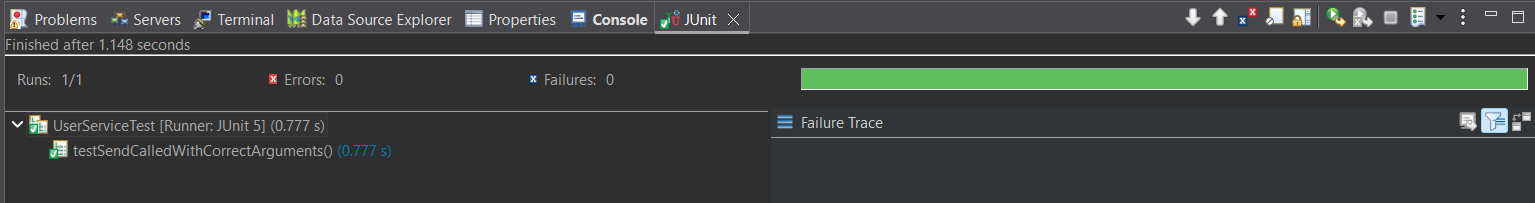
service.registerUser("user@example.com");

*verify*(mockNotification).send(*eq*("user@example.com"), *eq*("Welcome to our platform!"));

}

}

**Output:**

****

**Exercise 4: (Handling Void Methods)**

**Notifier.java:**

package demo2;

public class Notifier {

public void sendNotification() {

System.***out***.println("Notification sent!");

}

}

**NotifierTest.java:**

package demo2;

import static org.mockito.Mockito.*doNothing*;

import static org.mockito.Mockito.*mock*;

import static org.mockito.Mockito.*verify*;

import org.junit.jupiter.api.Test;

public class NotifierTest {

*@Test*

public void testSendNotification() {

Notifier notifier = *mock*(Notifier.class);

*doNothing*().when(notifier).sendNotification();

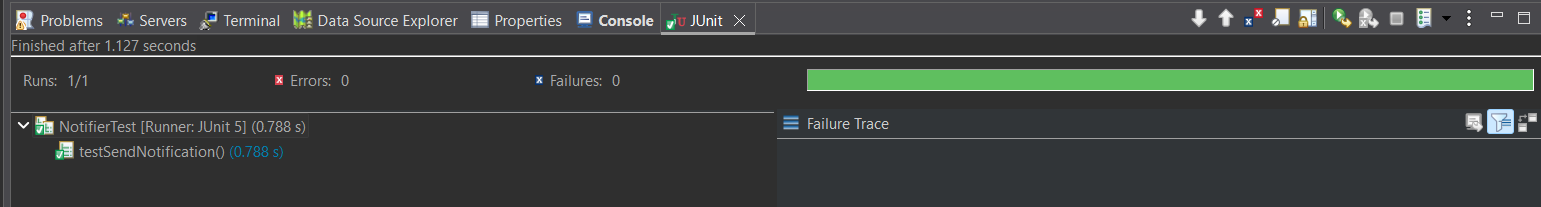
notifier.sendNotification();

*verify*(notifier).sendNotification();

}

}

**Output:**

****

**Exercise 5: (Mocking and Stubbing with Multiple Returns)**

**MyServiceTest.java:**

package demo2;

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

import static org.mockito.Mockito.*mock*;

import static org.mockito.Mockito.*times*;

import static org.mockito.Mockito.*verify*;

import static org.mockito.Mockito.*when*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

*@Test*

public void testMultipleReturnValues() {

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

*when*(mockApi.getData()).thenReturn("Value 1", "Value 2", "Value 3");

MyService service = new MyService(mockApi);

*assertEquals*("Value 1", service.fetchData());

*assertEquals*("Value 2", service.fetchData());

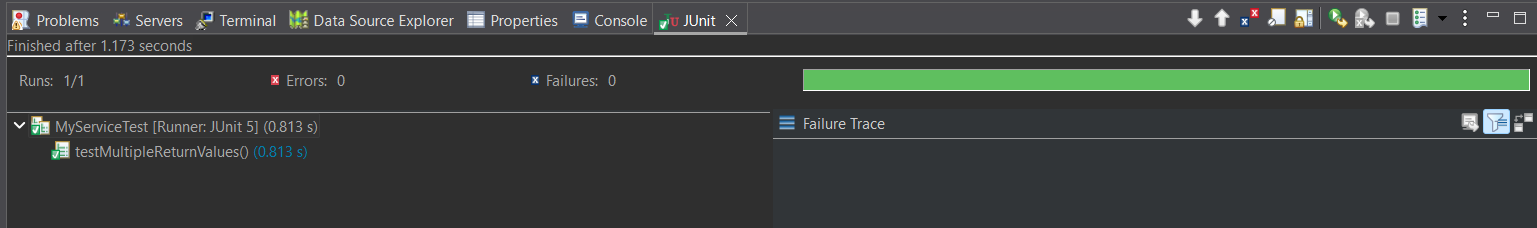
*assertEquals*("Value 3", service.fetchData());

*verify*(mockApi, *times*(3)).getData();

}

}

**Output:**

****

**Exercise 6: (Verifying Interaction Order)**

**ExternalApi2.java:**

package demo2;

public interface ExternalApi2 {

void connect();

void fetchData();

void disconnect();

}

**Workflow.java:**

package demo2;

public class Workflow {

private ExternalApi2 api;

public Workflow(ExternalApi2 api) {

this.api = api;

}

public void execute() {

api.connect();

api.fetchData();

api.disconnect();

}

}

**WorkflowTest.java:**

package demo2;

import static org.mockito.Mockito.*mock*;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

import org.mockito.Mockito;

public class WorkflowTest {

*@Test*

public void testMethodCallOrder() {

// Step 1: Create a mock object

ExternalApi2 mockApi = *mock*(ExternalApi2.class);

Workflow workflow = new Workflow(mockApi);

workflow.execute();

// Step 3: Verify the order of interaction

InOrder inOrder = Mockito.*inOrder*(mockApi);

inOrder.verify(mockApi).connect();

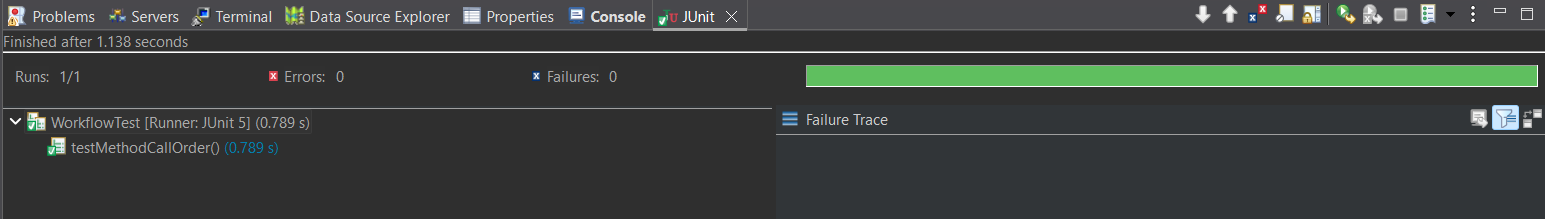
inOrder.verify(mockApi).fetchData();

inOrder.verify(mockApi).disconnect();

}

}

**Output:**

****

**Exercise 7: (Handling Void Methods with Exceptions)**

**NotifierTest.java:**

package demo2;

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

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

import static org.mockito.Mockito.*doThrow*;

import static org.mockito.Mockito.*mock*;

import org.junit.jupiter.api.Test;

public class NotifierTest {

*@Test*

void testSendNotification\_ThrowsException() {

Notifier notifier = *mock*(Notifier.class);

*doThrow*(new RuntimeException("Sending failed")).when(notifier).sendNotification();

RuntimeException exception = *assertThrows*(RuntimeException.class, () -> {

notifier.sendNotification();

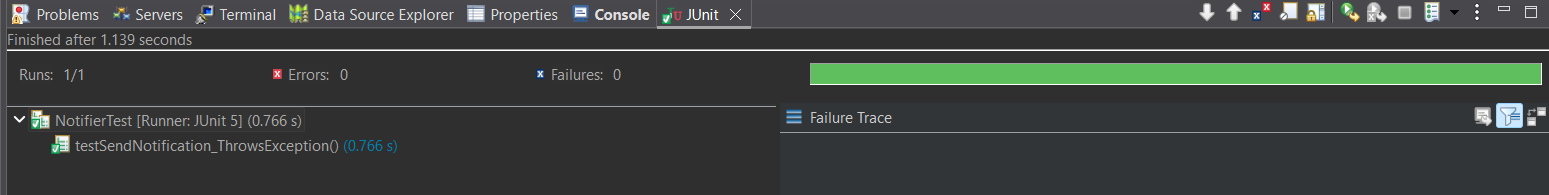
});

*assertEquals*("Sending failed", exception.getMessage());

}

}

**Output:**

****