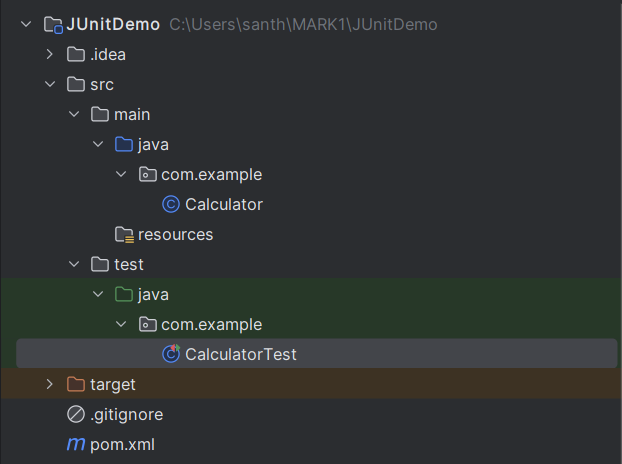
**JUnit Testing Exercises**

**Exercise 1: Setting Up Junit**

**Project Structure:**

****

**Calculator.java**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java**

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

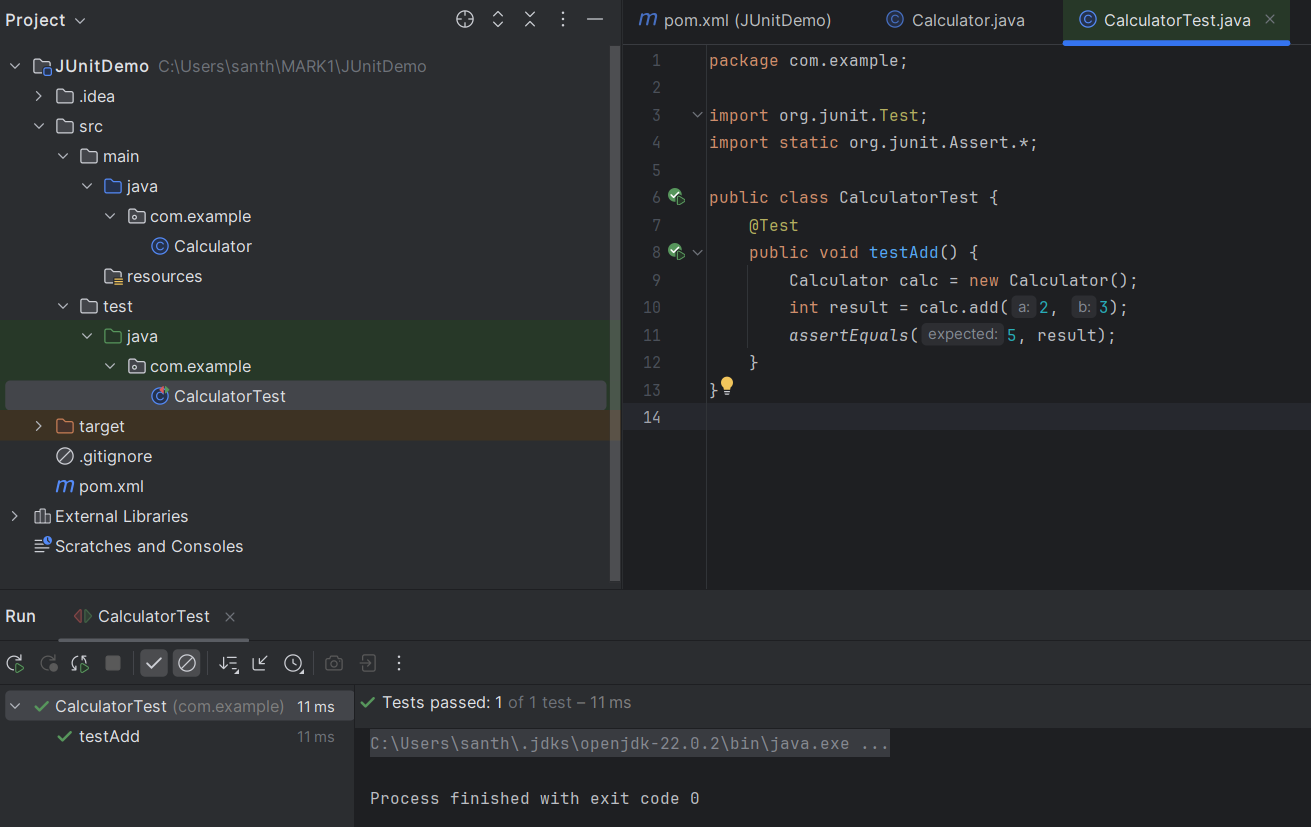
int result = calc.add(2, 3);

assertEquals(5, result);

}

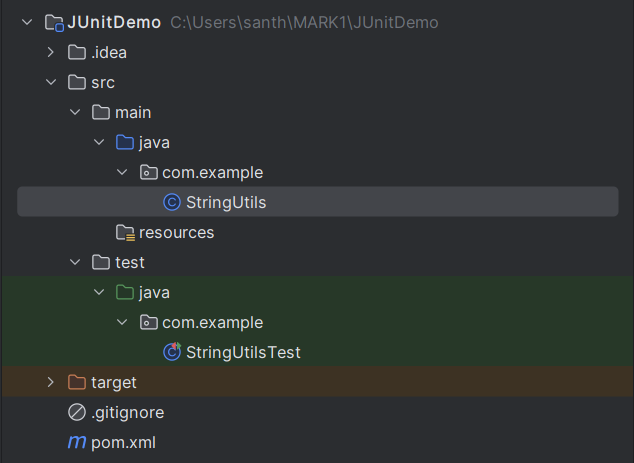
}

**Output:**

****

**Exercise 2: Writing Basic JUnit Tests**

**Project Structure:**

****

**StringUtils.java**

package com.example;

public class StringUtils {

public boolean isPalindrome(String str) {

if (str == null) return false;

String reversed = new StringBuilder(str).reverse().toString();

return str.equalsIgnoreCase(reversed);

}

public String reverse(String str) {

if (str == null) return null;

return new StringBuilder(str).reverse().toString();

}

public int countVowels(String str) {

if (str == null) return 0;

int count = 0;

for (char ch : str.toLowerCase().toCharArray()) {

if ("aeiou".indexOf(ch) != -1) {

count++;

}

}

return count;

}

}

**StringUtilsTest.java**

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class StringUtilsTest {

StringUtils utils = new StringUtils();

@Test

public void testIsPalindrome() {

assertTrue(utils.isPalindrome("Madam"));

assertFalse(utils.isPalindrome("Hello"));

}

@Test

public void testReverse() {

assertEquals("olleh", utils.reverse("hello"));

assertEquals("avaJ", utils.reverse("Java"));

}

@Test

public void testCountVowels() {

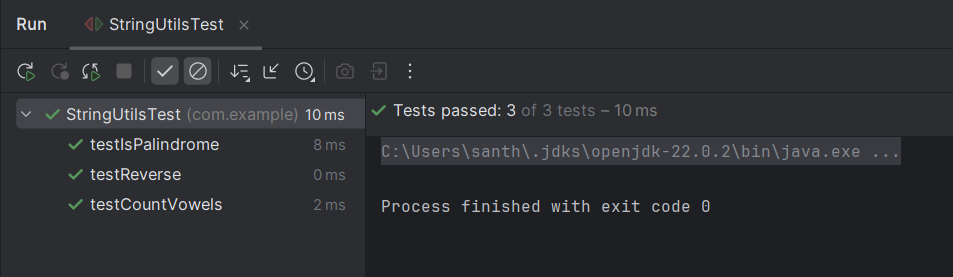
assertEquals(2, utils.countVowels("Hello"));

assertEquals(5, utils.countVowels("education"));

}

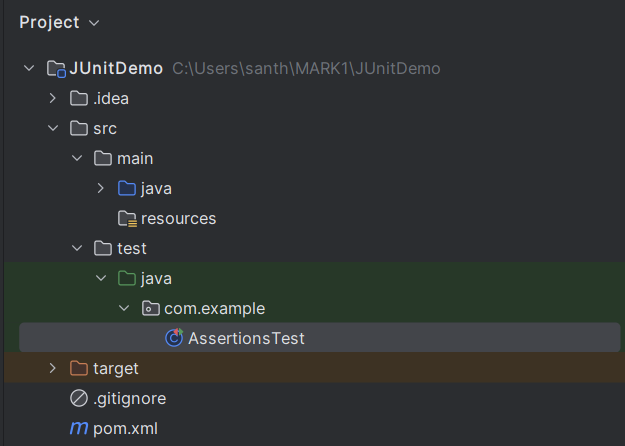
}

**Output:**

****

**Exercise 3: Assertions in Junit:**

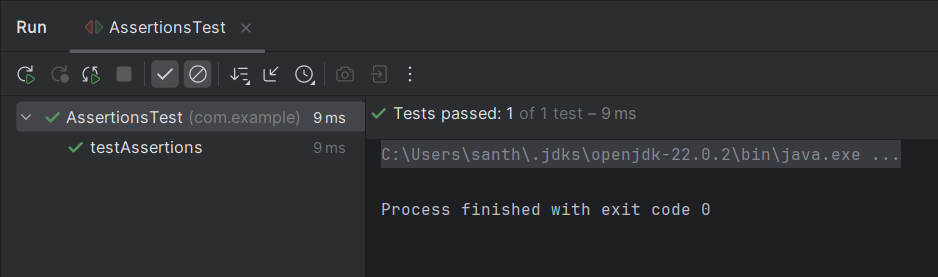
**Project Structure:**

****

**AssertionsTest.java**

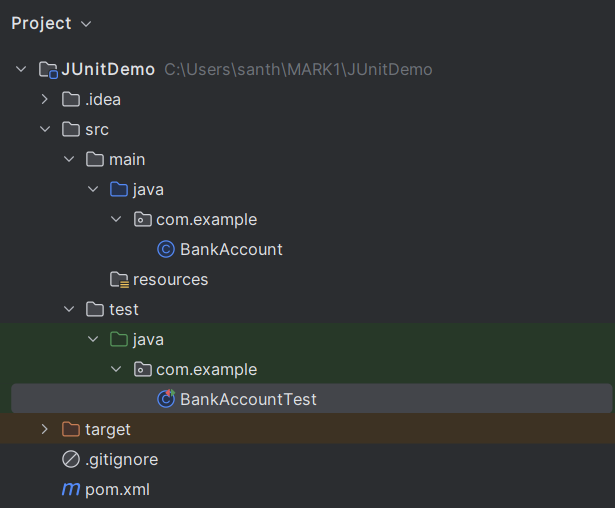
package com.example;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
public class AssertionsTest {  
 @Test  
 public void testAssertions() {  
 *assertEquals*(5, 2 + 3);  
 *assertTrue*(5 > 3);  
 *assertFalse*(5 < 3);  
 *assertNull*(null);  
 *assertNotNull*(new Object());  
 }  
}

**Output:**

****

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

**Project Structure:**

****

**BankAccount.java**

package com.example;  
public class BankAccount {  
 private int balance;  
 public BankAccount(int initialBalance) {  
 this.balance = initialBalance;  
 }  
 public void deposit(int amount) {  
 if (amount > 0)  
 balance += amount;  
 }  
 public void withdraw(int amount) {  
 if (amount > 0 && amount <= balance)  
 balance -= amount;  
 }  
 public int getBalance() {  
 return balance;  
 }  
}

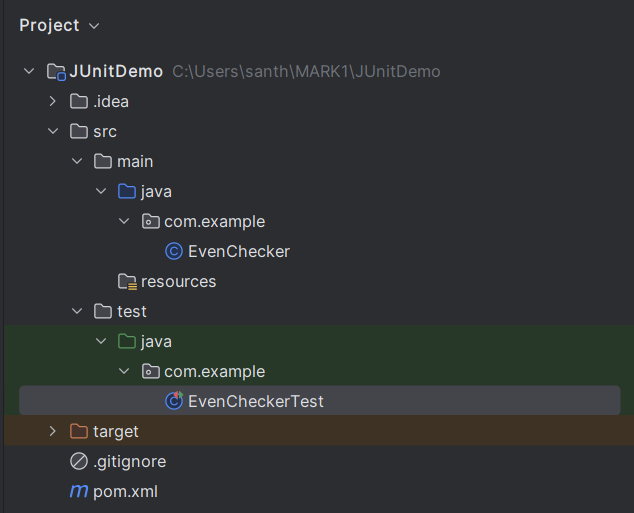
**BankAccountTest.java**

package com.example;  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
public class BankAccountTest {  
 private BankAccount account;  
  
 @Before  
 public void setUp() {  
 account = new BankAccount(100); // Initial balance  
 }  
  
 @After  
 public void tearDown() {  
 account = null;  
 }  
  
 @Test  
 public void testDeposit() {  
 int depositAmount = 50;  
 account.deposit(depositAmount);  
 *assertEquals*(150, account.getBalance());  
 }  
  
 @Test  
 public void testWithdraw() {  
 int withdrawAmount = 30;  
 account.withdraw(withdrawAmount);  
 *assertEquals*(70, account.getBalance());  
 }  
}

**Advanced JUnit Testing Exercises**

**Exercise 1: Parameterized Tests**

**Project Structure:**

****

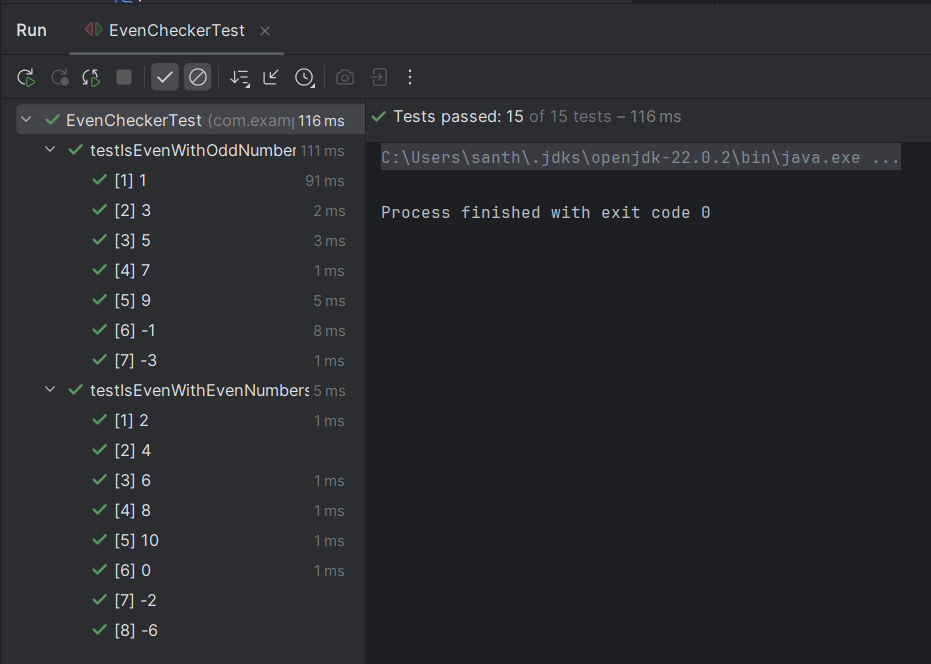
**EvenChecker.java**

package com.example;  
  
public class EvenChecker {  
 public boolean isEven(int number) {  
 return number % 2 == 0;  
 }  
}

**EvenCheckerTest.java**

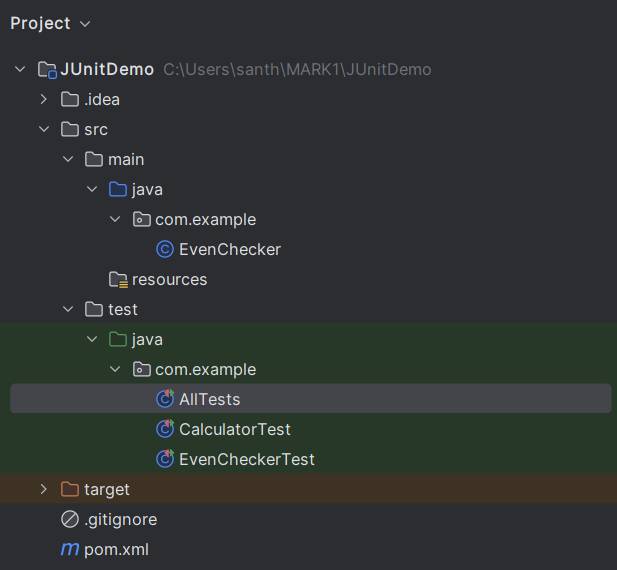
package com.example;  
  
import org.junit.jupiter.params.ParameterizedTest;  
import org.junit.jupiter.params.provider.ValueSource;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class EvenCheckerTest {  
 EvenChecker checker = new EvenChecker();  
  
 @ParameterizedTest  
 @ValueSource(ints = {2, 4, 6, 8, 10, 0, -2, -6})  
 public void testIsEvenWithEvenNumbers(int number) {  
 *assertTrue*(checker.isEven(number));  
 }  
  
 @ParameterizedTest  
 @ValueSource(ints = {1, 3, 5, 7, 9, -1, -3})  
 public void testIsEvenWithOddNumbers(int number) {  
 *assertFalse*(checker.isEven(number));  
 }  
}

**Output:**

****

**Exercise 2: Test Suites and Categories**

**Project Structure:**

****

**EvenChecker:**

package com.example;  
  
public class EvenChecker {  
 public boolean isEven(int number) {  
 return number % 2 == 0;  
 }  
}

**CalculatorTest.java**

package com.example;  
  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class CalculatorTest {  
 @Test  
 void testAdd() {  
 *assertEquals*(5, 2 + 3);  
 }  
}

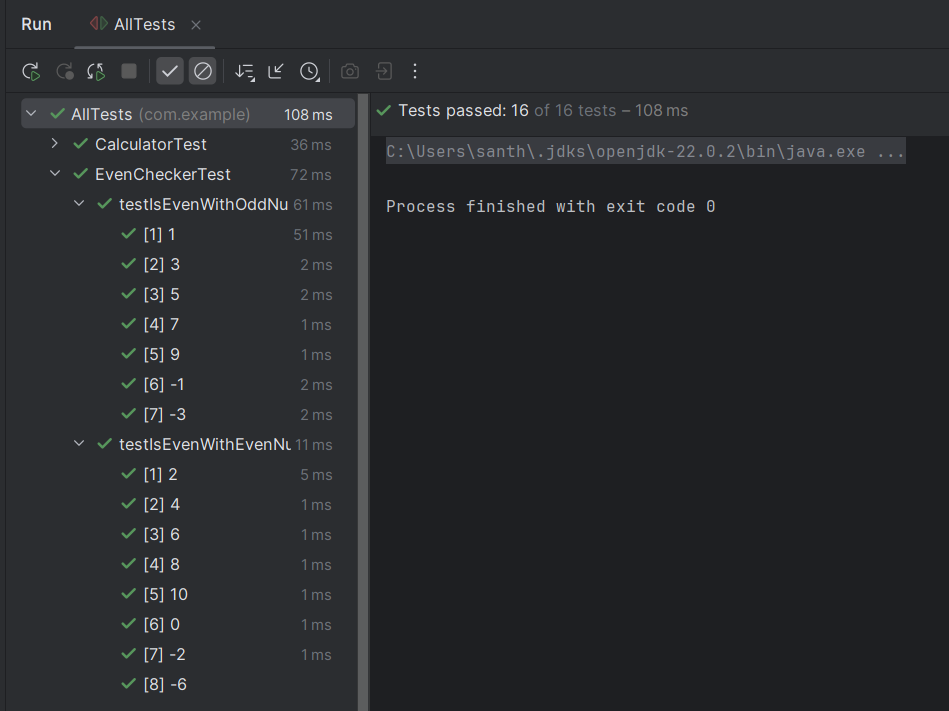
**EvencheckerTest.java**

package com.example;  
  
import org.junit.jupiter.params.ParameterizedTest;  
import org.junit.jupiter.params.provider.ValueSource;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class EvenCheckerTest {  
 EvenChecker checker = new EvenChecker();  
  
 @ParameterizedTest  
 @ValueSource(ints = {2, 4, 6, 8, 10, 0, -2, -6})  
 public void testIsEvenWithEvenNumbers(int number) {  
 *assertTrue*(checker.isEven(number));  
 }  
  
 @ParameterizedTest  
 @ValueSource(ints = {1, 3, 5, 7, 9, -1, -3})  
 public void testIsEvenWithOddNumbers(int number) {  
 *assertFalse*(checker.isEven(number));  
 }  
}

**AllTests.java**

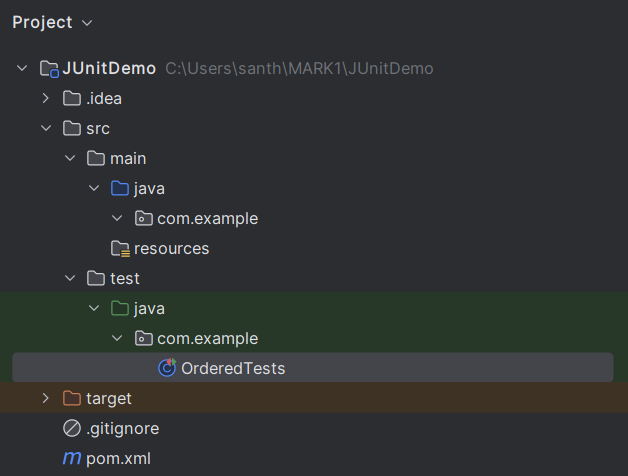
package com.example;  
  
import org.junit.platform.suite.api.SelectClasses;  
import org.junit.platform.suite.api.Suite;  
  
@Suite  
@SelectClasses({  
 CalculatorTest.class,  
 EvenCheckerTest.class  
})  
public class AllTests {  
}

**Output:**

****

**Exercise 3: Test Execution Order**

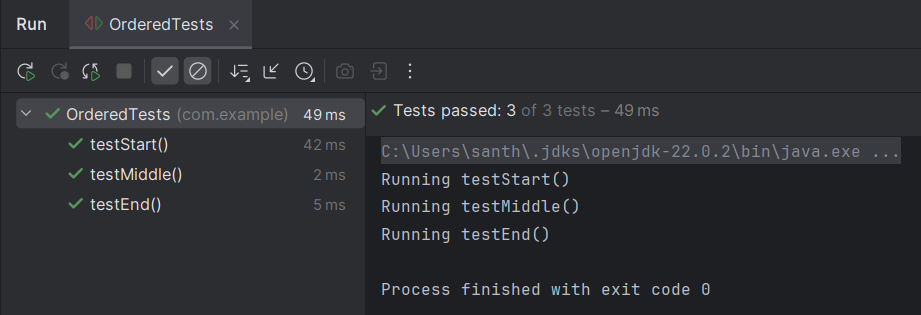
**Project Structure:**

****

**OrderedTests.java**

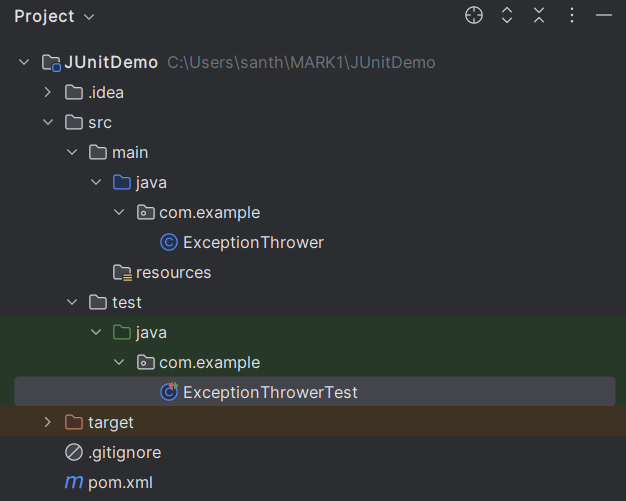
package com.example;  
  
import org.junit.jupiter.api.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)  
public class OrderedTests {  
 static StringBuilder *executionLog* = new StringBuilder();  
  
 @Test  
 @Order(1)  
 void testStart() {  
 *executionLog*.append("Start->");  
 System.*out*.println("Running testStart()");  
 *assertTrue*(true);  
 }  
  
 @Test  
 @Order(2)  
 void testMiddle() {  
 *executionLog*.append("Middle->");  
 System.*out*.println("Running testMiddle()");  
 *assertTrue*(true);  
 }  
  
 @Test  
 @Order(3)  
 void testEnd() {  
 *executionLog*.append("End");  
 System.*out*.println("Running testEnd()");  
 *assertEquals*("Start->Middle->End", *executionLog*.toString());  
 }  
}

**Output:**

****

**Exercise 4: Exception Testing**

**Project Structure:**

****

**ExceptionThrower.java**

package com.example;  
  
public class ExceptionThrower {  
 public void throwException(String input) {  
 if (input == null || input.isEmpty()) {  
 throw new IllegalArgumentException("Input cannot be null or empty");  
 }  
 System.*out*.println("Input is valid: " + input);  
 }  
}

**ExceptionThrowerTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

public class ExceptionThrowerTest {

@Test

public void testThrowExceptionWithInvalidInput() {

ExceptionThrower thrower = new ExceptionThrower();

// Expecting IllegalArgumentException

Exception exception = assertThrows(IllegalArgumentException.class, () -> {

thrower.throwException("");

});

assertEquals("Input cannot be null or empty", exception.getMessage());

}

@Test

public void testThrowExceptionWithValidInput() {

ExceptionThrower thrower = new ExceptionThrower();

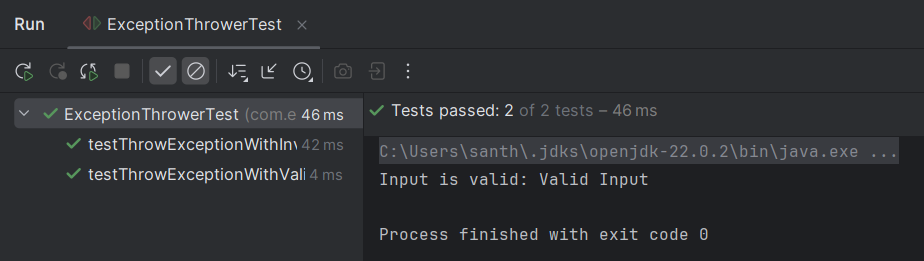
// No exception should be thrown

assertDoesNotThrow(() -> thrower.throwException("Valid Input"));

}

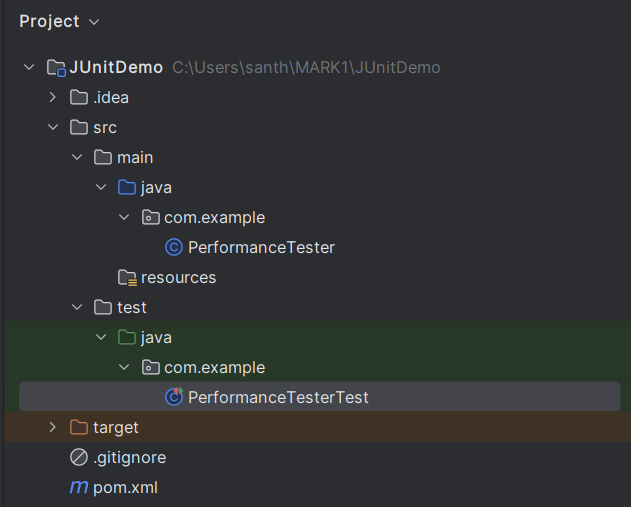
}

**Output:**

****

**Exercise 5: Timeout and Performance Testing**

**Project Structure:**

****

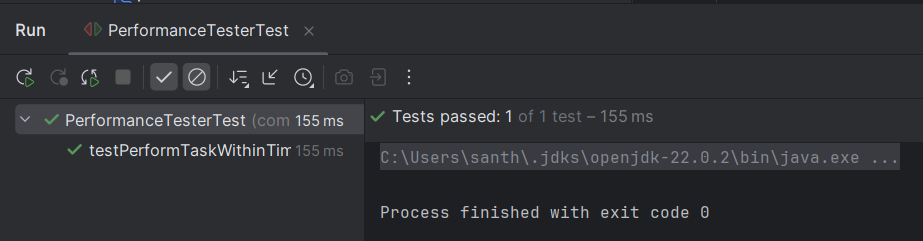
**PerformanceTester.java**

package com.example;  
  
public class PerformanceTester {  
  
 public void performTask() {  
 try {  
 Thread.*sleep*(100);   
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
}

**PerformanceTesterTest.java**

package com.example;  
  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.*assertTimeout*;  
import java.time.Duration;  
public class PerformanceTesterTest {  
  
 @Test  
 public void testPerformTaskWithinTimeout() {  
 PerformanceTester tester = new PerformanceTester();  
 *assertTimeout*(Duration.*ofMillis*(200), tester::performTask);  
 }  
}

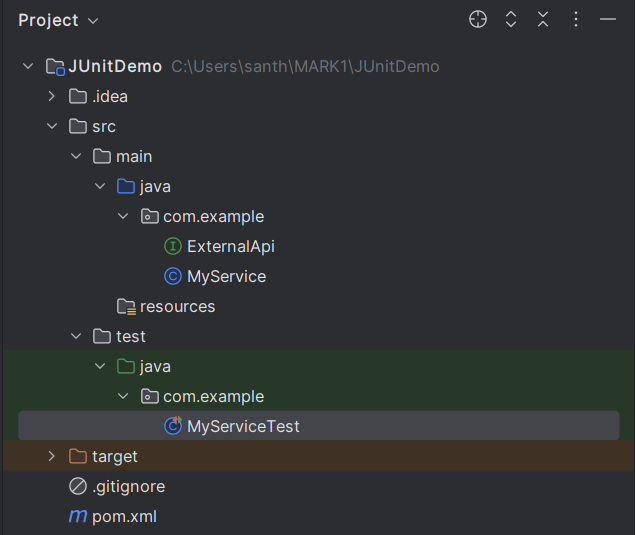
**Output:**

****

**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

**Project Structure:**

****

**ExternalApi.java**

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

**MyService.java**

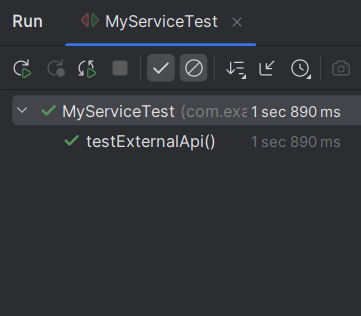
package com.example;  
  
public class MyService {  
 private final ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData();  
 }  
}

**MyServiceTest.java**

package com.example;import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;

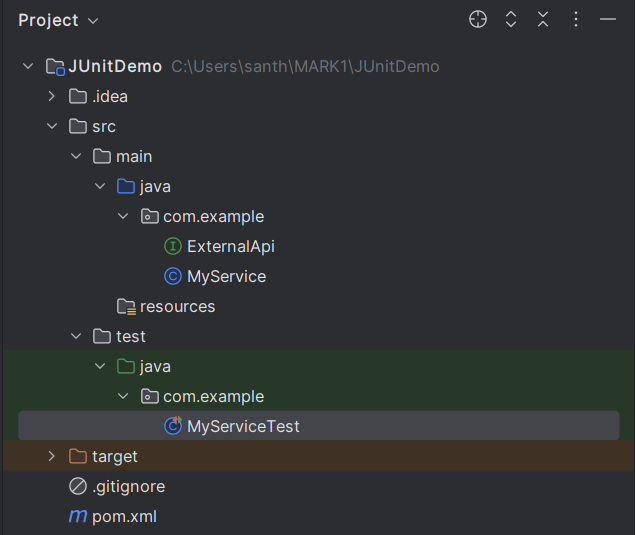
import static org.junit.jupiter.api.Assertions.\*;  
public class MyServiceTest {  
  
 @Test  
 public void testExternalApi() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 *when*(mockApi.getData()).thenReturn("Mock Data");  
  
 MyService service = new MyService(mockApi);  
 String result = service.fetchData();  
  
 *assertEquals*("Mock Data", result);  
 }  
}

**Output:**

****

**Exercise 2: Verifying Interactions**

**Project Structure:**

****

**ExternalApi.java**

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

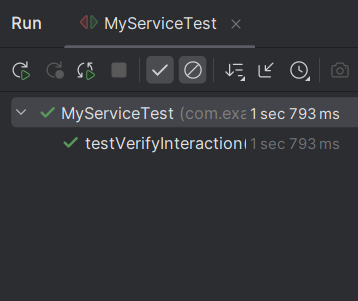
**MyService.java**

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

**MyServiceTest.java**

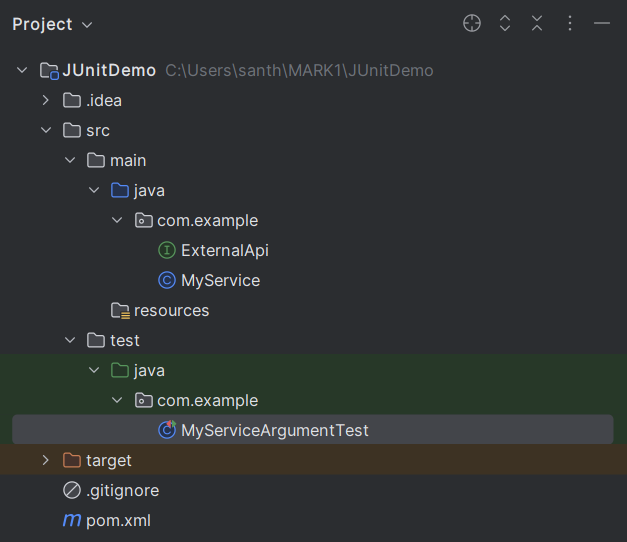
package com.example;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testVerifyInteraction() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 MyService service = new MyService(mockApi);  
  
 service.fetchData();  
  
 *verify*(mockApi).getData();  
 }  
}

**Output:**

****

**Exercise 3: Argument Matching**

**Project Structure:**

****

**ExternalApi.java**

package com.example;  
  
public interface ExternalApi {  
 String getData();  
 String getDataById(String id);  
}

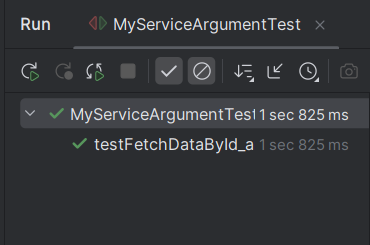
**MyServcie.java**

package com.example;  
public class MyService {  
 private final ExternalApi api;   
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchDataById(String id) {  
 return api.getDataById(id);  
 }  
}

**MyServiceArgument.java**

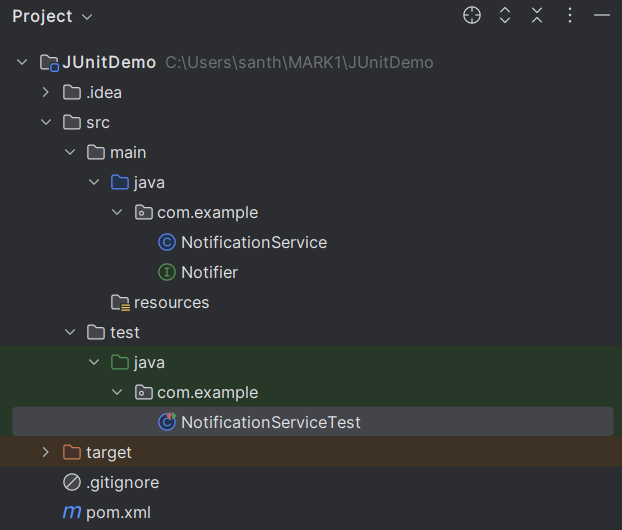
package com.example;  
  
import org.junit.jupiter.api.Test;  
import org.mockito.ArgumentCaptor;  
  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceArgumentTest {  
  
 @Test  
 void testFetchDataById\_argumentMatching() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
  
 *when*(mockApi.getDataById(*eq*("42"))).thenReturn("The Answer");  
  
 MyService service = new MyService(mockApi);  
 String result = service.fetchDataById("42");  
  
 *verify*(mockApi).getDataById(*eq*("42"));  
 *assertEquals*("The Answer", result);  
  
 ArgumentCaptor<String> captor = ArgumentCaptor.*forClass*(String.class);  
 *verify*(mockApi).getDataById(captor.capture());  
 *assertEquals*("42", captor.getValue());  
 }  
}

**Output:**

****

**Exercise 4: Handling Void Methods**

**Project Structure:**

****

**Notifier.java**

package com.example;  
  
public interface Notifier {  
 void send(String message);  
}

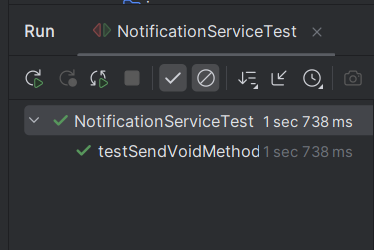
**NotificationService.java**

package com.example;  
public class NotificationService {  
 private final Notifier notifier;  
  
 public NotificationService(Notifier notifier) {  
 this.notifier = notifier;  
 }  
  
 public void alert(String message) {  
 // Internally calls the void method  
 notifier.send(message);  
 }  
}

**NotificationServiceTest.java**

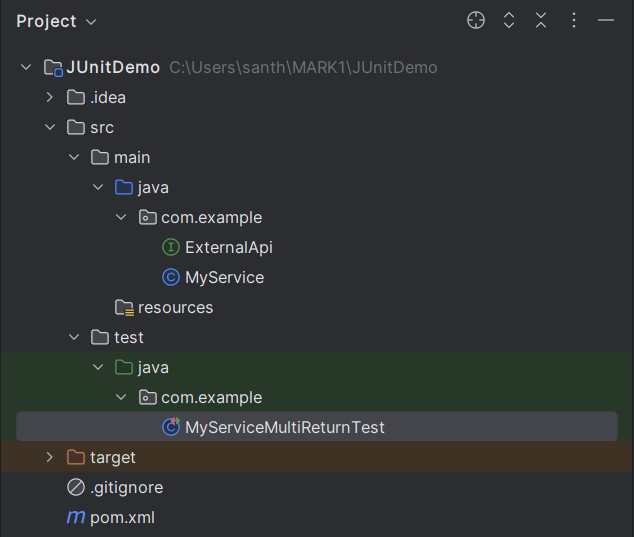
package com.example;  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
  
public class NotificationServiceTest {  
  
 @Test  
 public void testSendVoidMethodInteraction() {  
 Notifier mockNotifier = *mock*(Notifier.class);  
  
 *doNothing*().when(mockNotifier).send("Hello");  
  
 NotificationService service = new NotificationService(mockNotifier);  
 service.alert("Hello");  
  
 *verify*(mockNotifier, *times*(1)).send("Hello");  
 }  
}

**Output:**

****

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

**Project Structure:**

****

**ExternalApi.java**

package com.example;  
  
public interface ExternalApi {  
 String getStatus();  
}

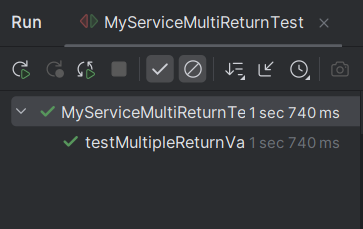
**MyService.java**

package com.example;  
public class MyService {  
 private final ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String checkStatus() {  
 return api.getStatus();  
 }  
}

**MyServiceMultiReturnTest.java**

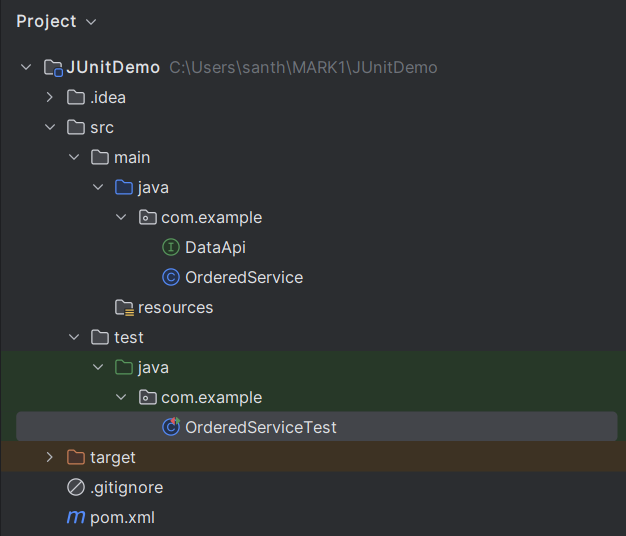
package com.example;  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
public class MyServiceMultiReturnTest {  
  
 @Test  
 public void testMultipleReturnValues() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 *when*(mockApi.getStatus())  
 .thenReturn("LOADING")  
 .thenReturn("PROCESSING")  
 .thenReturn("COMPLETED");  
 MyService service = new MyService(mockApi);  
  
 *assertEquals*("LOADING", service.checkStatus());  
 *assertEquals*("PROCESSING", service.checkStatus());  
 *assertEquals*("COMPLETED", service.checkStatus());  
 }  
}

**Output:**

****

**Exercise 6: Verifying Interaction Order**

**Project Structure:**

****

**DataApi.java**

package com.example;

public interface DataApi {

void connect();

void fetchData();

void disconnect();

}

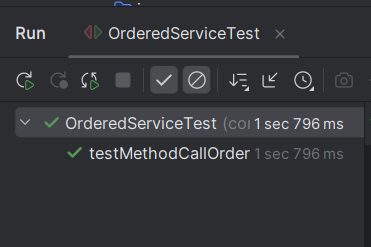
**OrderedService.java**

package com.example;  
public class OrderedService {  
  
 private final DataApi api;  
  
 public OrderedService(DataApi api) {  
 this.api = api;  
 }  
  
 public void process() {  
 api.connect();  
 api.fetchData();  
 api.disconnect();  
 }  
}

**OrderedServcieTest.java**

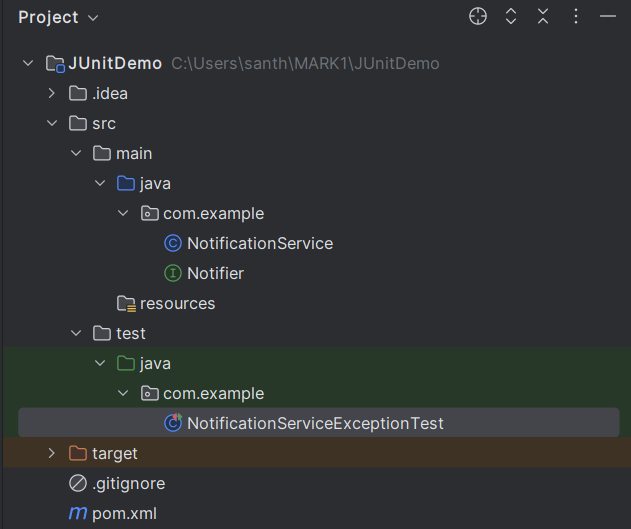
package com.example;  
import org.junit.jupiter.api.Test;  
import org.mockito.InOrder;  
import static org.mockito.Mockito.\*;  
public class OrderedServiceTest {  
  
 @Test  
 public void testMethodCallOrder() {  
 DataApi mockApi = *mock*(DataApi.class);  
   
 OrderedService service = new OrderedService(mockApi);  
  
 service.process();  
   
 InOrder inOrder = *inOrder*(mockApi);  
 inOrder.verify(mockApi).connect();  
 inOrder.verify(mockApi).fetchData();  
 inOrder.verify(mockApi).disconnect();  
 }  
}

**Output:**

****

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

**Project Structure:**

****

**Notifier.java**

package com.example;  
public interface Notifier {  
 void send(String message) throws RuntimeException;  
}

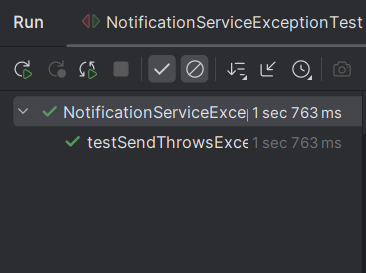
**NotificationService.java**

package com.example;  
public class NotificationService {  
 private final Notifier notifier;  
  
 public NotificationService(Notifier notifier) {  
 this.notifier = notifier;  
 }  
  
 public void alert(String message) {  
 notifier.send(message);  
 }  
}

**NotificationServiceExceptionTest.java**

package com.example;  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.Mockito.\*;  
public class NotificationServiceExceptionTest {  
  
 @Test  
 public void testSendThrowsException() {  
 Notifier mockNotifier = *mock*(Notifier.class);  
   
 *doThrow*(new RuntimeException("Send failed")).when(mockNotifier).send("CRITICAL");  
   
 NotificationService service = new NotificationService(mockNotifier);  
   
 RuntimeException thrown = *assertThrows*(RuntimeException.class, () -> {  
 service.alert("CRITICAL");  
 });  
  
 *assertEquals*("Send failed", thrown.getMessage());  
   
 *verify*(mockNotifier).send("CRITICAL");  
 }  
}

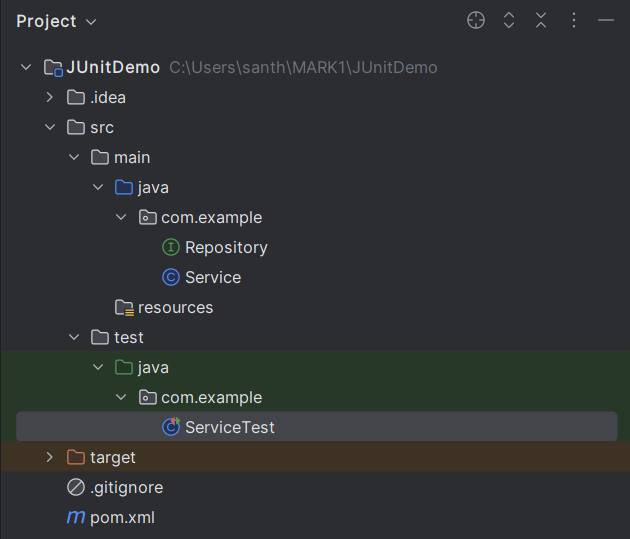
**Output:**

****

**Advanced Mockito Hands-On Exercises**

**Exercise 1: Mocking Databases and Repositories**

**Project Structure:**

****

**Repository.java**

package com.example;  
public interface Repository {  
 String getData();   
}

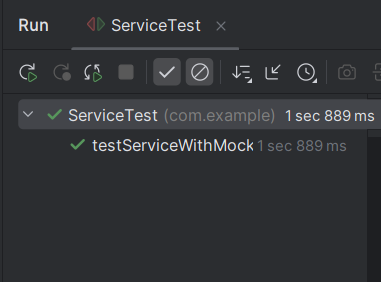
**Service.java**

package com.example;  
public class Service {  
 private final Repository repository;  
  
 public Service(Repository repository) {  
 this.repository = repository;  
 }  
  
 public String processData() {  
 // Example logic: fetch data and add prefix  
 String data = repository.getData();  
 return "Processed " + data;  
 }  
}

**ServiceTest.java**

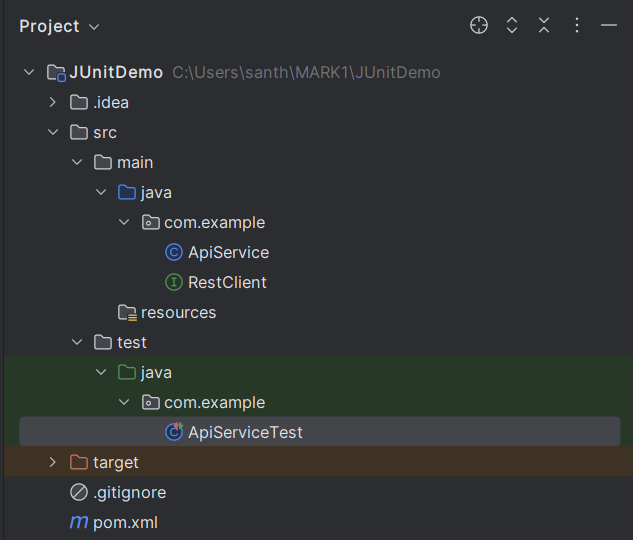
package com.example;  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
public class ServiceTest {  
  
 @Test  
 public void testServiceWithMockRepository() {  
 Repository mockRepository = *mock*(Repository.class);  
   
 *when*(mockRepository.getData()).thenReturn("Mock Data");  
   
 Service service = new Service(mockRepository);  
 String result = service.processData();  
   
 *assertEquals*("Processed Mock Data", result);  
 *verify*(mockRepository).getData();   
 }  
}

**Output**



**Exercise 2: Mocking External Services (RESTful APIs)**

**Project Structure:**

****

**RestClient.java**

package com.example;  
  
public interface RestClient {  
 String getResponse();   
}

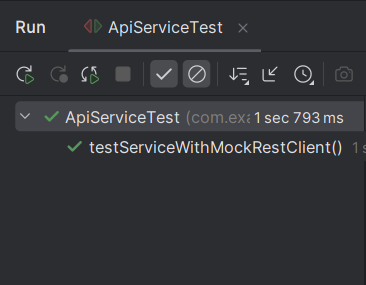
**ApiService.java**

package com.example;  
public class ApiService {  
 private final RestClient restClient;  
  
 public ApiService(RestClient restClient) {  
 this.restClient = restClient;  
 }  
  
 public String fetchData() {  
 return "Fetched " + restClient.getResponse();  
 }  
}

**ApiServcieTest.java**

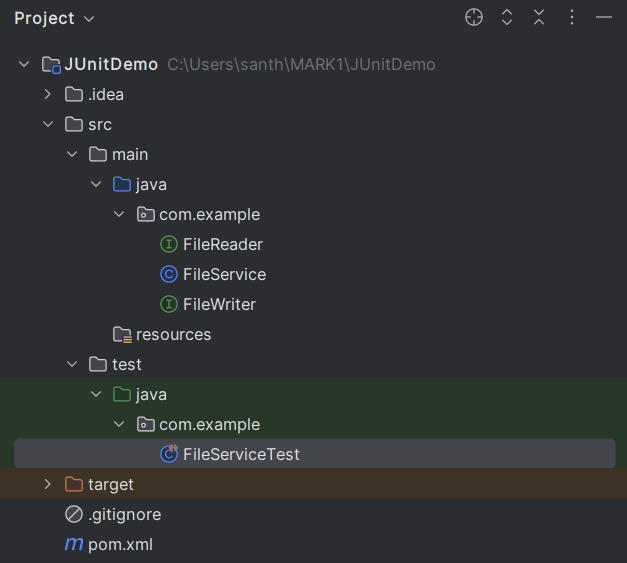
package com.example;  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
public class ApiServiceTest {  
  
 @Test  
 public void testServiceWithMockRestClient() {  
 RestClient mockRestClient = *mock*(RestClient.class);  
   
 *when*(mockRestClient.getResponse()).thenReturn("Mock Response");  
   
 ApiService apiService = new ApiService(mockRestClient);  
 String result = apiService.fetchData();  
   
 *assertEquals*("Fetched Mock Response", result);  
 *verify*(mockRestClient).getResponse();  
 }  
}

**Output:**

****

**Exercise 3: Mocking File I/O**

**Project structure:**

****

**FileReader.java**

package com.example;

public interface FileReader {

String read();

}

**FileWriter.java**

package com.example;  
public interface FileWriter {  
 void write(String content);  
}

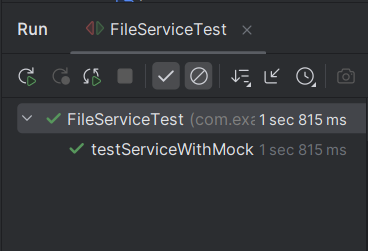
**FileService.java**

package com.example;  
public class FileService {  
 private final FileReader fileReader;  
 private final FileWriter fileWriter;  
  
 public FileService(FileReader fileReader, FileWriter fileWriter) {  
 this.fileReader = fileReader;  
 this.fileWriter = fileWriter;  
 }  
 public String processFile() {  
 String content = fileReader.read();  
 String processed = "Processed " + content;  
 fileWriter.write(processed);  
 return processed;  
 }  
}

**FileServiceTest.java**

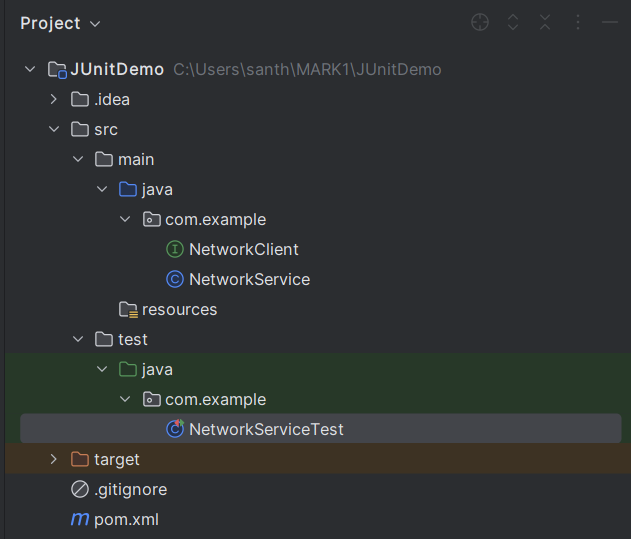
package com.example;  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
public class FileServiceTest {  
  
 @Test  
 public void testServiceWithMockFileIO() {  
 FileReader mockFileReader = *mock*(FileReader.class);  
 FileWriter mockFileWriter = *mock*(FileWriter.class);  
   
 *when*(mockFileReader.read()).thenReturn("Mock File Content");  
   
 FileService fileService = new FileService(mockFileReader, mockFileWriter);  
 String result = fileService.processFile();  
   
 *assertEquals*("Processed Mock File Content", result);  
   
 *verify*(mockFileReader).read();  
 *verify*(mockFileWriter).write("Processed Mock File Content");  
 }  
}

**Output:**

****

**Exercise 4: Mocking Network Interactions**

**Project Structure:**

****

**NetworkClient.java**

package com.example;  
public interface NetworkClient {  
 String connect();   
}

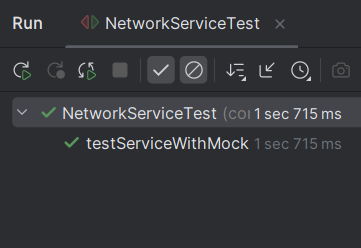
**NetworkService.java**

package com.example;  
public class NetworkService {  
 private final NetworkClient networkClient;  
  
 public NetworkService(NetworkClient networkClient) {  
 this.networkClient = networkClient;  
 }  
  
 public String connectToServer() {  
 return "Connected to " + networkClient.connect();  
 }  
}

**NetworkServiceTest.java**

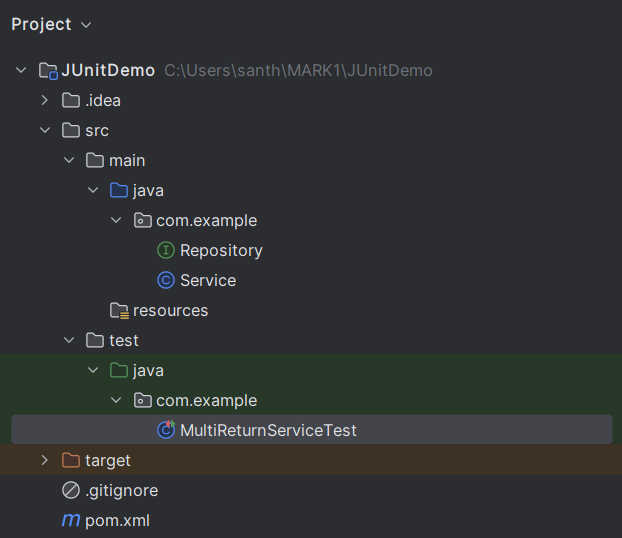
package com.example;  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
public class NetworkServiceTest {  
  
 @Test  
 public void testServiceWithMockNetworkClient() {  
 NetworkClient mockNetworkClient = *mock*(NetworkClient.class);  
   
 *when*(mockNetworkClient.connect()).thenReturn("Mock Connection");  
   
 NetworkService networkService = new NetworkService(mockNetworkClient);  
   
 String result = networkService.connectToServer();  
 *assertEquals*("Connected to Mock Connection", result);  
  
 *verify*(mockNetworkClient).connect();  
 }  
}

**Output:**

****

**Exercise 5: Mocking Multiple Return Values**

**Project Structure:**

****

**Repository.java**

package com.example;  
public interface Repository {  
 String getData();  
}

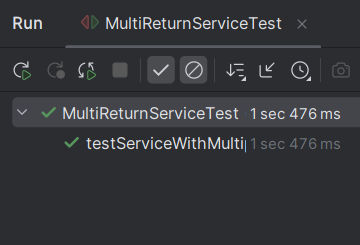
**Service.java**

package com.example;  
public class Service {  
 private final Repository repository;  
  
 public Service(Repository repository) {  
 this.repository = repository;  
 }  
  
 public String processData() {  
 String data = repository.getData();  
 return "Processed " + data;  
 }  
}

**MultiReturnServiceTest.java**

package com.example;  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
public class MultiReturnServiceTest {  
  
 @Test  
 public void testServiceWithMultipleReturnValues() {  
 Repository mockRepository = *mock*(Repository.class);  
   
 *when*(mockRepository.getData())  
 .thenReturn("First Mock Data")  
 .thenReturn("Second Mock Data");  
   
 Service service = new Service(mockRepository);  
   
 String firstResult = service.processData();  
 String secondResult = service.processData();  
  
 *assertEquals*("Processed First Mock Data", firstResult);  
 *assertEquals*("Processed Second Mock Data", secondResult);  
   
 *verify*(mockRepository, *times*(2)).getData();  
 }  
}

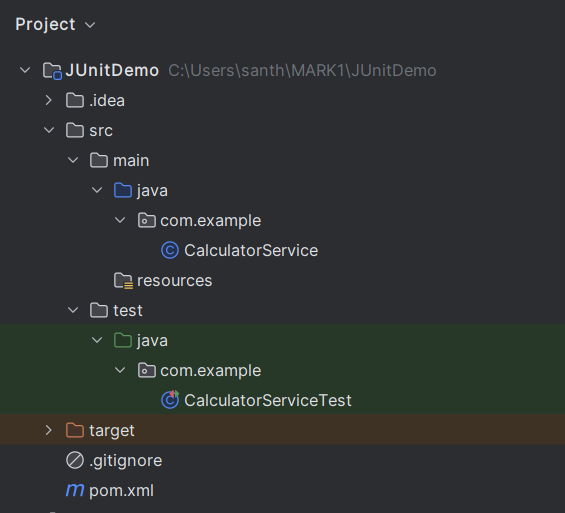
**Output:**

****

**Spring Testing Exercises**

**Exercise 1: Basic Unit Test for a Service Method**

**Project Structure:**

****

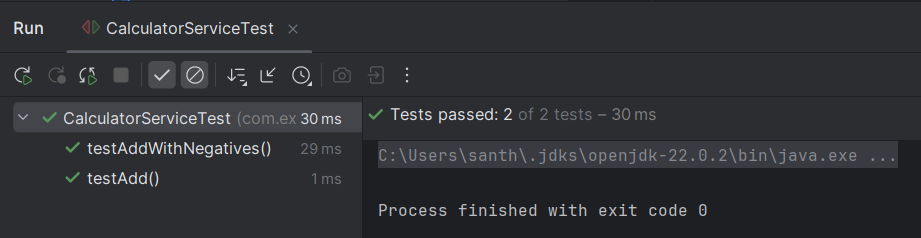
**CalculatorService.java**

package com.example;  
import org.springframework.stereotype.Service;  
  
@Service  
public class CalculatorService {  
 public int add(int a, int b) {  
 return a + b;  
 }  
}

**CalculatorServiceTest.java**

package com.example;  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
public class CalculatorServiceTest {  
  
 @Test  
 public void testAdd() {  
 CalculatorService calculatorService = new CalculatorService();  
   
 int result = calculatorService.add(10, 20);  
   
 *assertEquals*(30, result, "Addition of 10 and 20 should be 30");  
 }  
 @Test  
 public void testAddWithNegatives() {  
 CalculatorService calculatorService = new CalculatorService();  
 *assertEquals*(-5, calculatorService.add(-2, -3));  
 *assertEquals*(3, calculatorService.add(5, -2));  
 }  
}

**Output:**

****

**Exercise 2: Mocking a Repository in a Service Test**

**Project Structure:**

**User.java**

package com.example.demo.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class User {

@Id

private Long id;

private String name;

// Constructors

public User() {}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

// Getters & Setters

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

**UserRepository.java**

package com.example.demo.repository;

import com.example.demo.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

package com.example.demo.service;

import com.example.demo.model.User;

import com.example.demo.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public User getUserById(Long id) {

return userRepository.findById(id).orElse(null);

}

}

**UserServiceTest.java**

package com.example.demo.service;

import com.example.demo.model.User;

import com.example.demo.repository.UserRepository;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import java.util.Optional;

import static org.mockito.Mockito.\*;

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

public class UserServiceTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

public UserServiceTest() {

MockitoAnnotations.openMocks(this); // Initialize mocks

}

@Test

public void testGetUserById() {

// Arrange

Long id = 1L;

User mockUser = new User(id, "Santhosh");

when(userRepository.findById(id)).thenReturn(Optional.of(mockUser));

// Act

User result = userService.getUserById(id);

// Assert

assertNotNull(result);

assertEquals("Santhosh", result.getName());

verify(userRepository).findById(id);

}

}

**Exercise 3: Testing a REST Controller with MockMvc**

**User.java**

package com.example.demo.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class User {

@Id

private Long id;

private String name;

public User() {}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

public Long getId() { return id; }

public String getName() { return name; }

public void setId(Long id) { this.id = id; }

public void setName(String name) { this.name = name; }

}

**UserController.java**

package com.example.demo.controller;

import com.example.demo.model.User;

import com.example.demo.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserService userService;

@GetMapping("/{id}")

public ResponseEntity<User> getUser(@PathVariable Long id) {

return ResponseEntity.ok(userService.getUserById(id));

}

}

**UserService.java**

package com.example.demo.service;

import com.example.demo.model.User;

import org.springframework.stereotype.Service;

@Service

public class UserService {

public User getUserById(Long id) {

return new User(id, "Default");

}

}

**UserControllerTest.java**

package com.example.demo.controller;

import com.example.demo.model.User;

import com.example.demo.service.UserService;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.boot.test.mock.mockito.MockBean;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.test.web.servlet.MockMvc;

import static org.mockito.Mockito.\*;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

import org.springframework.http.MediaType;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;

@WebMvcTest(UserController.class)

public class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

public void testGetUserById() throws Exception {

// Arrange

User mockUser = new User(1L, "Santhosh");

when(userService.getUserById(1L)).thenReturn(mockUser);

// Act & Assert

mockMvc.perform(get("/users/1"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.id").value(1L))

.andExpect(jsonPath("$.name").value("Santhosh"));

}

}