1. **JUnit\_Basic Testing Exercises:**

**Exercise 1: Setting Up JUnit:**

Step 1: Create a New Java Project using Maven

mvn archetype:generate -DgroupId=com.example -DartifactId=JUnitDemo -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

Step 2: Add JUnit Dependency in pom.xml

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

Step 3: Create a New Test Class

**Hello.java:**

package com.example;

public class Hello {

public String sayHello() {

return "Hello, JUnit!";

}

}

**HelloTest.java:**

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class HelloTest {

@Test

public void testSayHello() {

Hello hello = new Hello();

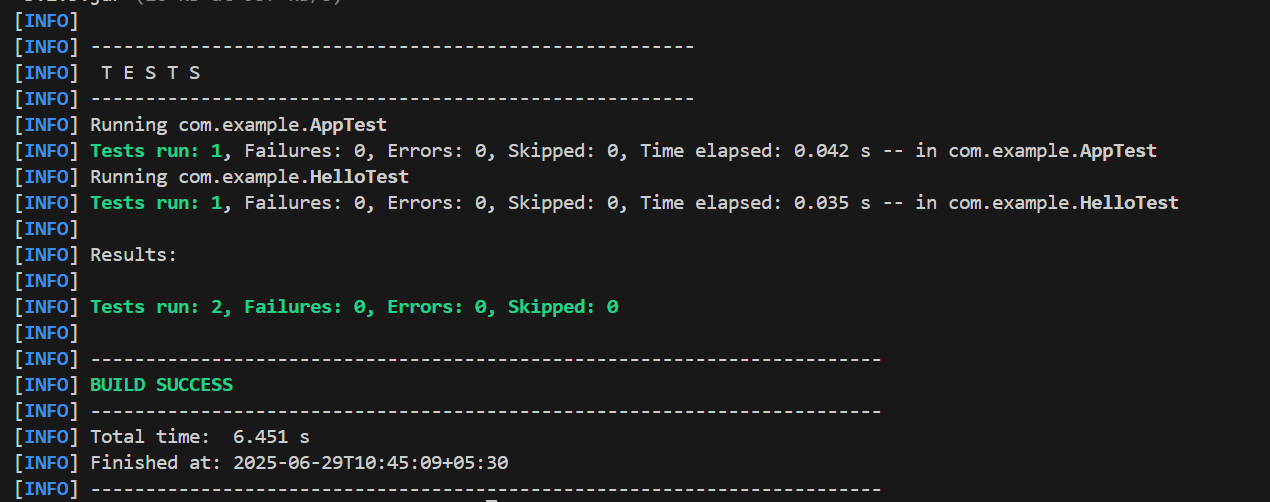
String result = hello.sayHello();

assertEquals("Hello, JUnit!", result);

}

}

**Output:**



**Exercise 2: Writing Basic JUnit Tests:**

**Calculator.java**

package com.example;

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("Division by zero not allowed");

}

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();

assertEquals(10, calc.add(6, 4));

}

@Test

public void testSubtract() {

Calculator calc = new Calculator();

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

}

@Test

public void testMultiply() {

Calculator calc = new Calculator();

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

}

@Test

public void testDivide() {

Calculator calc = new Calculator();

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

}

@Test(expected = IllegalArgumentException.class)

public void testDivideByZero() {

Calculator calc = new Calculator();

calc.divide(10, 0); // should throw exception

}

}

**Output:**



**Exercise 3: Assertions in Junit:**

**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);

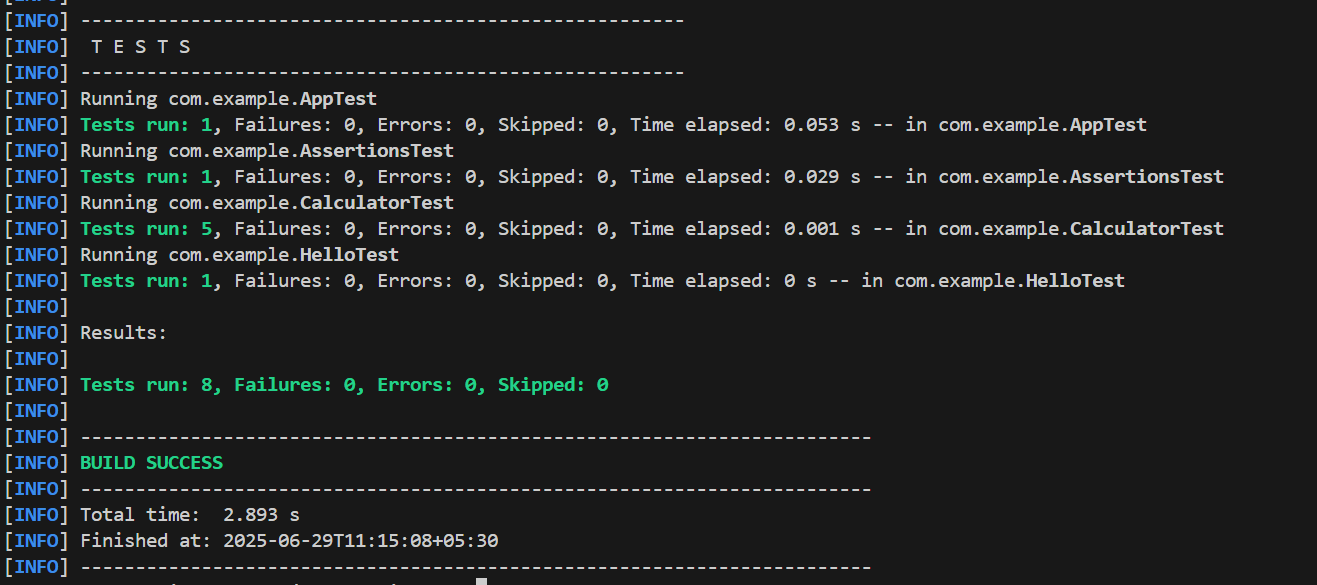
assertNull(null);

assertNotNull(new Object());

}

}

**Output:**



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

**CalculatorAAATest.java:**

package com.example;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorAAATest {

private Calculator calculator;

@Before

public void setUp() {

System.out.println("Setting up Calculator instance...");

calculator = new Calculator(); // Arrange

}

@After

public void tearDown() {

System.out.println("Cleaning up after test...\n");

calculator = null;

}

@Test

public void testAdd() {

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

assertEquals(5, result);

}

@Test

public void testSubtract() {

int result = calculator.subtract(7, 4);

assertEquals(3, result);

}

@Test

public void testMultiply() {

int result = calculator.multiply(4, 5);

assertEquals(20, result);

}

@Test

public void testDivide() {

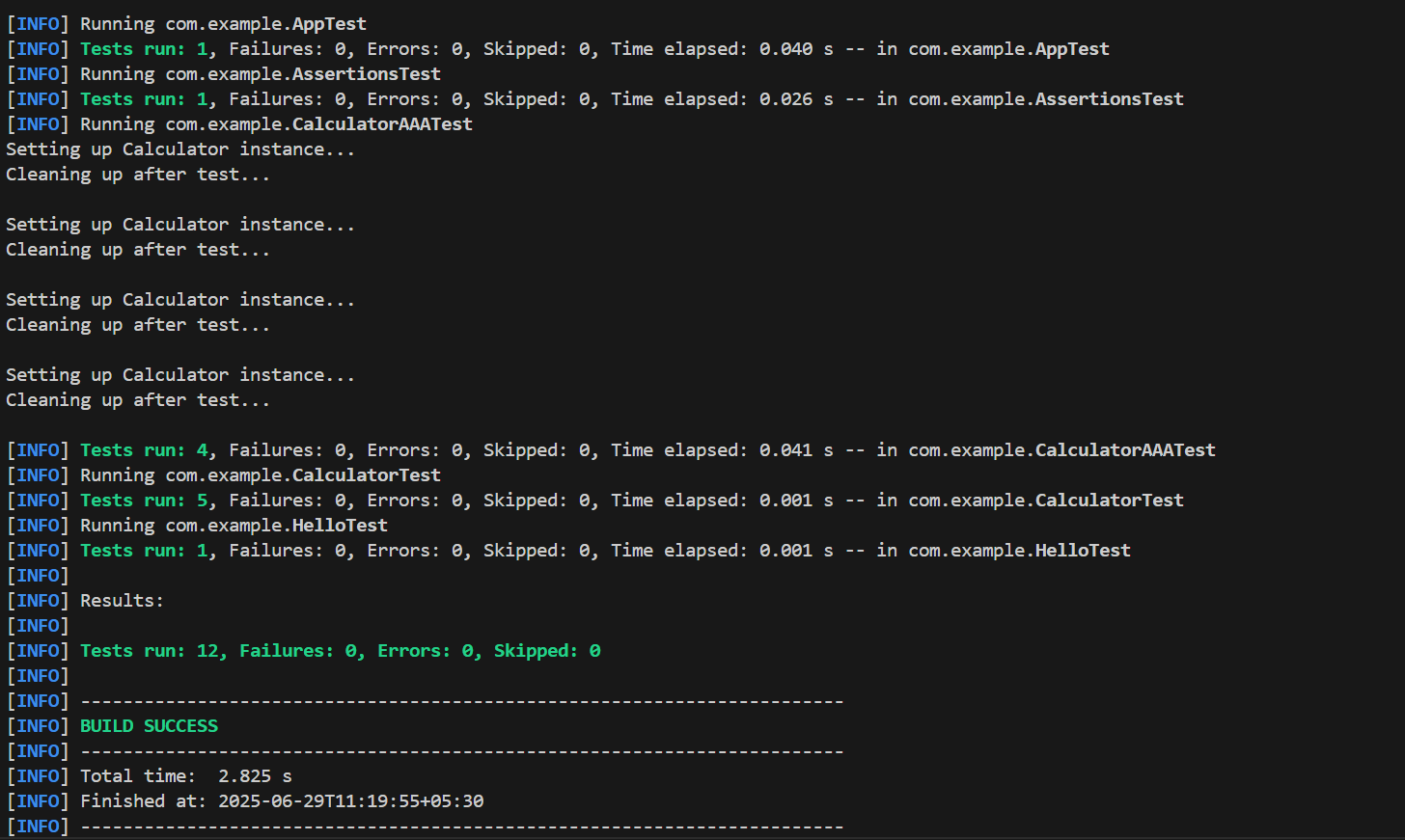
int result = calculator.divide(20, 4);

assertEquals(5, result);

}

}

**Output:**



**2.Advanced JUnit Testing Exercises:**

**Exercise 1: Parameterized Tests in JUnit 5:**

**Step 1: Update pom.xml for JUnit 5**

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.22.2</version>

</plugin>

</plugins>

</build>

**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 {

@ParameterizedTest

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

public void testIsEvenWithEvenNumbers(int number) {

EvenChecker checker = new EvenChecker();

assertTrue(checker.isEven(number));

}

@ParameterizedTest

@ValueSource(ints = {1, 3, 5, 7, 9})

public void testIsEvenWithOddNumbers(int number) {

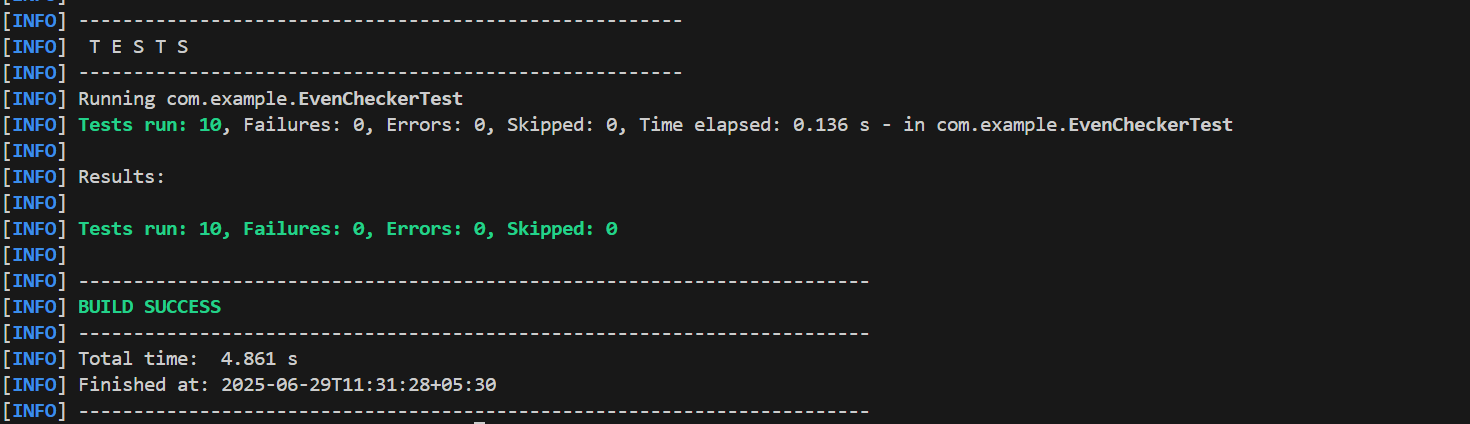
EvenChecker checker = new EvenChecker();

assertFalse(checker.isEven(number));

}

}

**Output:**

****

**Exercise 2: Test Suites and Categories:**

**Calculator.java**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**EvenChecker.java**

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.assertEquals;

public class CalculatorTest {

@Test

void testAddition() {

Calculator calc = new Calculator();

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

}

}

**EvenCheckerTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

public class EvenCheckerTest {

@Test

void testIsEven() {

EvenChecker checker = new EvenChecker();

assertTrue(checker.isEven(4));

}

}

**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 {

}

**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 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>JUnitDemo</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.platform</groupId>

<artifactId>junit-platform-suite-api</artifactId>

<version>1.10.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.1.2</version>

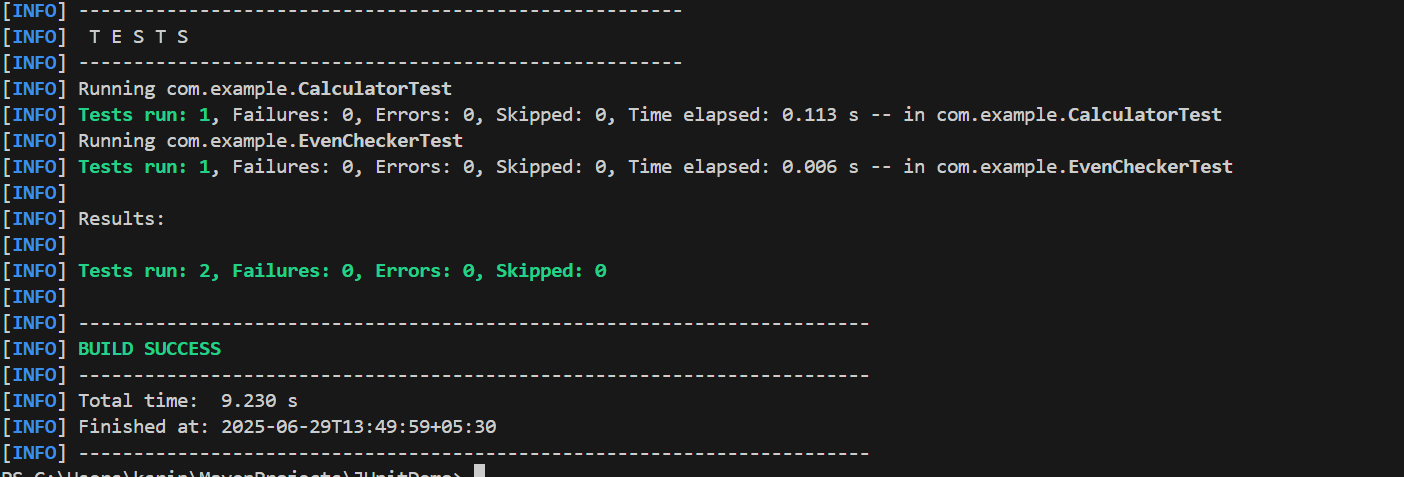
</plugin>

</plugins>

</build>

</project>

**Output:**

****

**Exercise 3: Test Execution Order:**

**OrderedTests.java**

package com.example;

import org.junit.jupiter.api.Order;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestMethodOrder;

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

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

@TestMethodOrder(OrderAnnotation.class)

public class OrderedTests {

@Test

@Order(2)

void testB() {

System.out.println("Running testB");

assertTrue(true);

}

@Test

@Order(1)

void testA() {

System.out.println("Running testA");

assertTrue(true);

}

@Test

@Order(3)

void testC() {

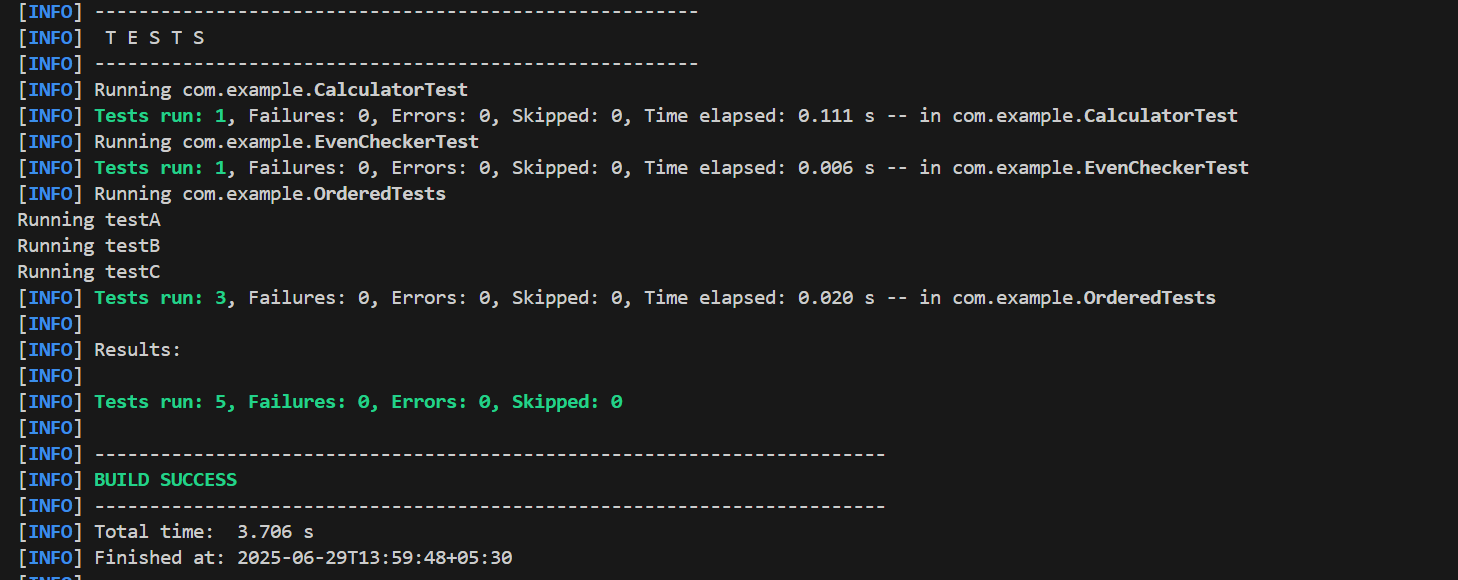
System.out.println("Running testC");

assertTrue(true);

}

}

**Output:**

****

**Exercise 4: Exception Testing:**

**ExceptionThrower.java**

package com.example;

public class ExceptionThrower {

public void throwException(boolean shouldThrow) {

if (shouldThrow) {

throw new IllegalArgumentException("Invalid input!");

}

}

}

**ExceptionThrowerTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

public class ExceptionThrowerTest {

@Test

public void testExceptionThrown() {

ExceptionThrower thrower = new ExceptionThrower();

assertThrows(IllegalArgumentException.class, () -> {

thrower.throwException(true);

});

}

@Test

public void testNoExceptionThrown() {

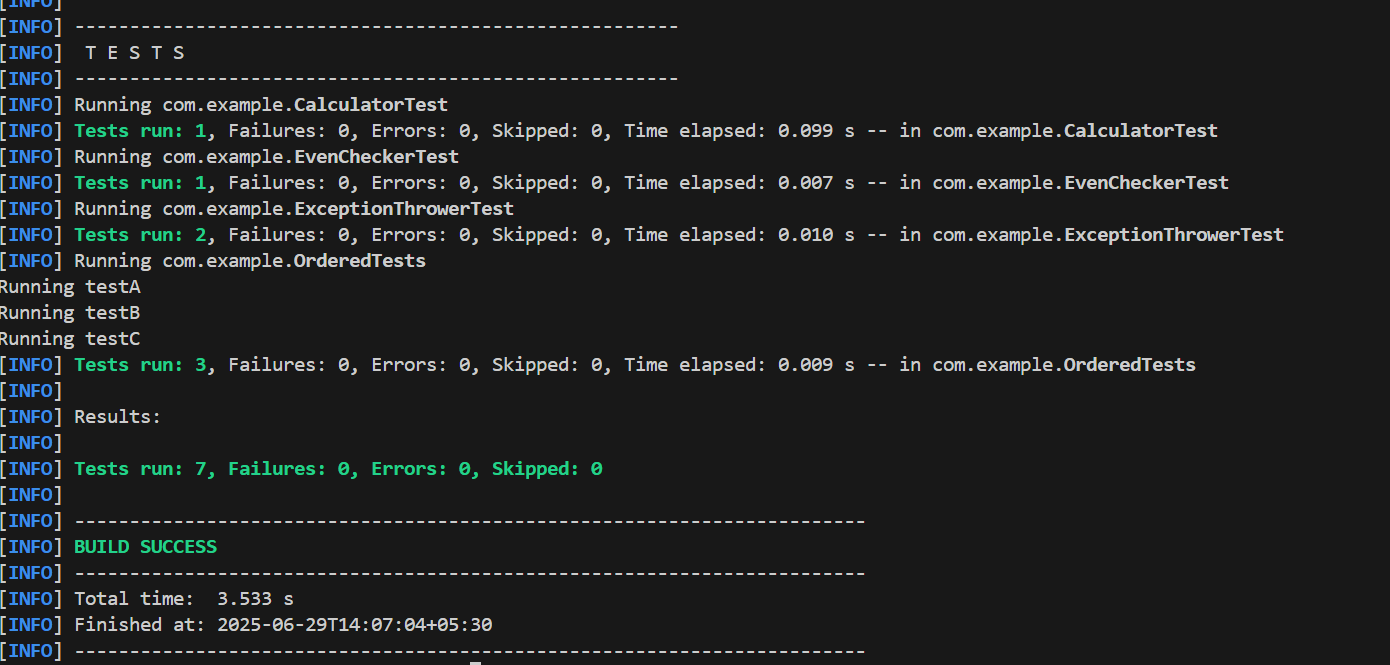
ExceptionThrower thrower = new ExceptionThrower();

thrower.throwException(false);

}

}

**Output:**

****

**Exercise 5: Timeout and Performance Testing**

**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 testPerformTaskWithinTimeLimit() {

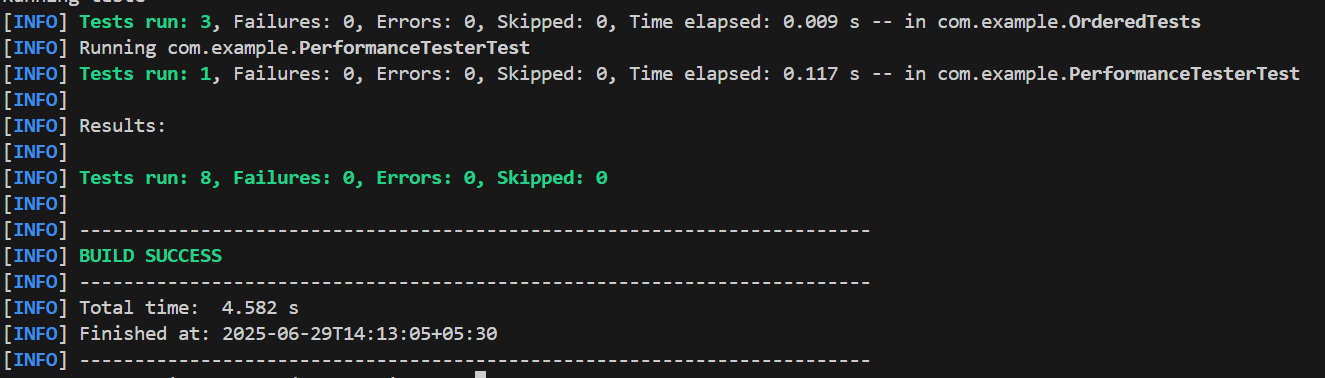
PerformanceTester tester = new PerformanceTester();

assertTimeout(Duration.ofMillis(200), tester::performTask);

}

}

**Output:**

****

**3.Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

**Step 1: Add Mockito Dependency**

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version>

<scope>test</scope>

</dependency>

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example;

public class MyService {

private 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.junit.jupiter.api.Assertions.assertEquals;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

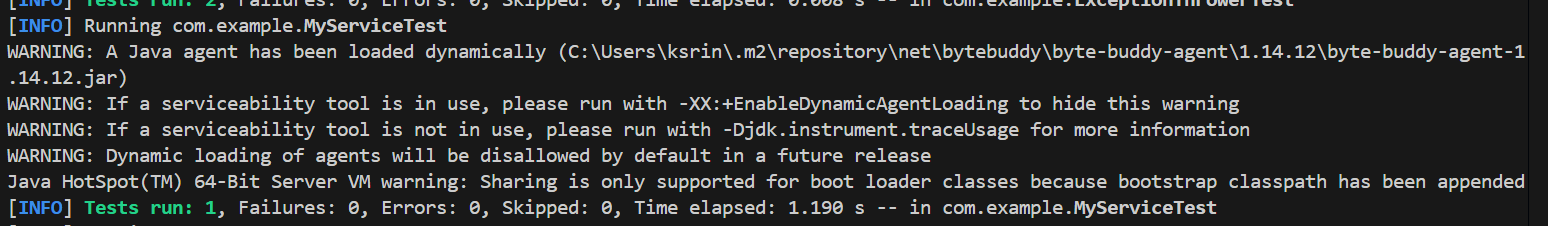
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**Output:**

****

**Exercise 2: Verifying Interactions**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

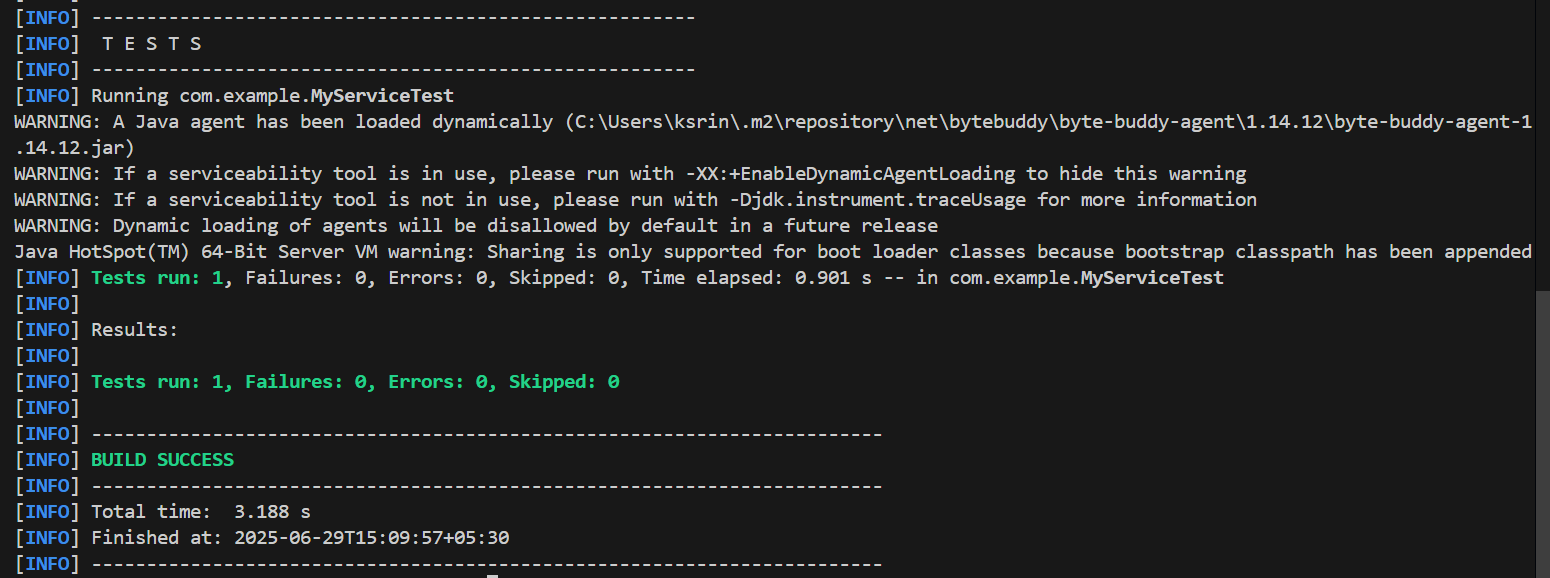
service.fetchData();

verify(mockApi).getData();

}

}

**Output:**

****

**Exercise 3: Argument Matching**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData(String key);

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData(String key) {

return api.getData(key);

}

}

**MyServiceTest.java**

package com.example;

import static org.mockito.Mockito.\*;

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

import static org.mockito.ArgumentMatchers.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testArgumentMatching() {

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

when(mockApi.getData(anyString())).thenReturn("Mocked Response");

MyService service = new MyService(mockApi);

String result = service.fetchData("exampleKey");

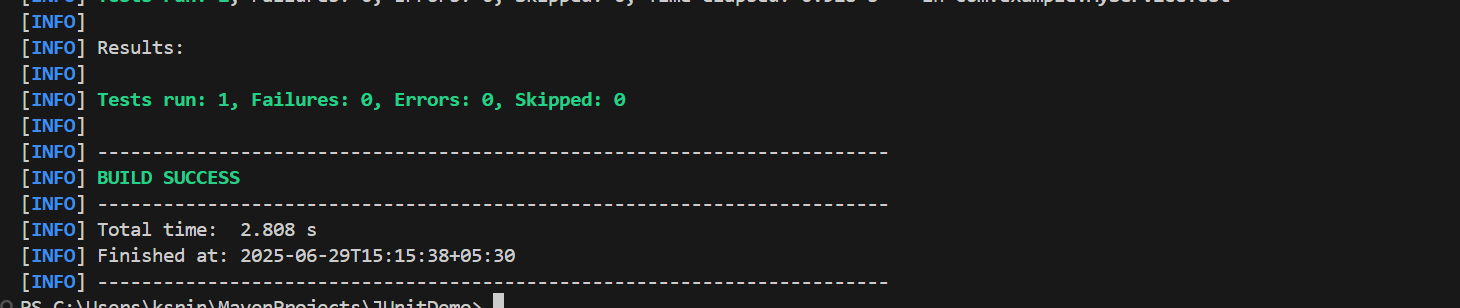
assertEquals("Mocked Response", result);

verify(mockApi).getData(eq("exampleKey"));

}

}

**Output:**

****

**Exercise 4: Handling Void Methods**

**Notifier.java**

package com.example;

public interface Notifier {

void sendNotification(String message);

}

**NotificationService.java**

package com.example;

public class NotificationService {

private Notifier notifier;

public NotificationService(Notifier notifier) {

this.notifier = notifier;

}

public void alert(String msg) {

notifier.sendNotification(msg);

}

}

**NotificationServiceTest.java**

package com.example;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class NotificationServiceTest {

@Test

public void testVoidMethod() {

Notifier mockNotifier = mock(Notifier.class);

doNothing().when(mockNotifier).sendNotification(anyString());

NotificationService service = new NotificationService(mockNotifier);

service.alert("Server down");

verify(mockNotifier).sendNotification("Server down");

}

}

**pom.xml**

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

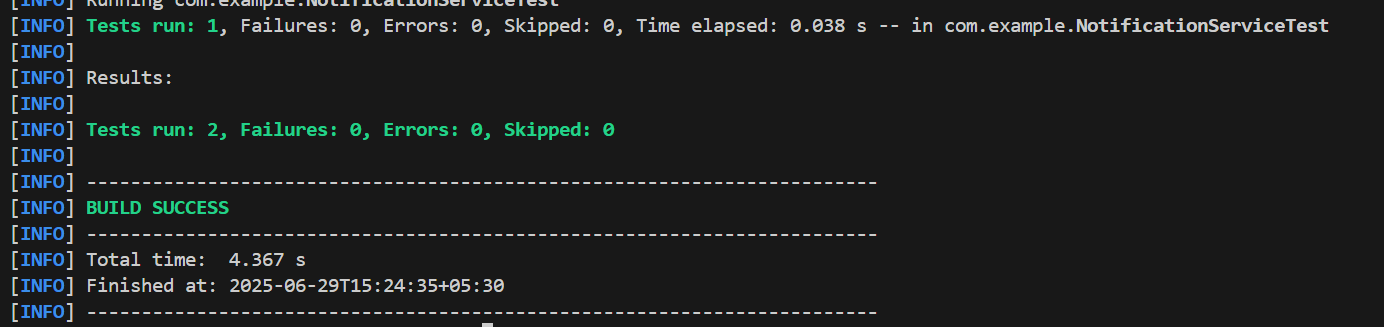
<version>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

**Output:**

****

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

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceMultipleReturnTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

public class MyServiceMultipleReturnTest {

@Test

public void testMultipleReturns() {

ExternalApi mockApi = mock(ExternalApi.class);

when(mockApi.getData())

.thenReturn("First")

.thenReturn("Second")

.thenReturn("Third");

MyService service = new MyService(mockApi);

assertEquals("First", service.fetchData());

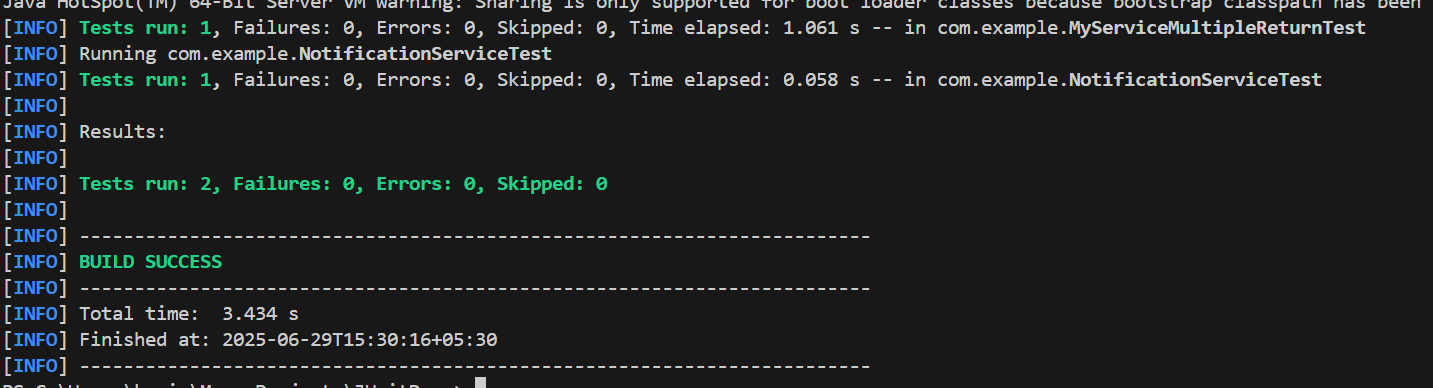
assertEquals("Second", service.fetchData());

assertEquals("Third", service.fetchData());

}

}

**Output:**

****

**Exercise 6: Verifying Interaction Order**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

void initialize();

void fetch();

void close();

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void process() {

api.initialize();

api.fetch();

api.close();

}

}

**MyServiceOrderTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

import org.mockito.InOrder;

public class MyServiceOrderTest {

@Test

public void testMethodCallOrder() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.process();

InOrder inOrder = inOrder(mockApi);

inOrder.verify(mockApi).initialize();

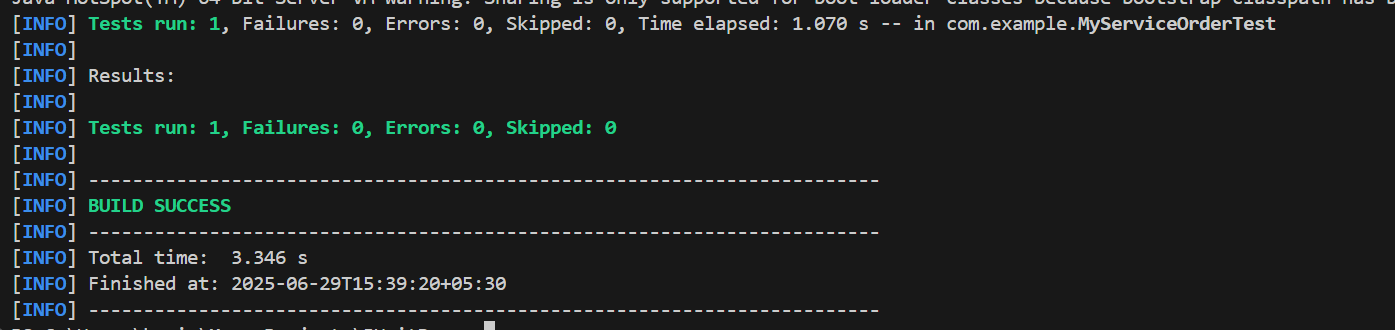
inOrder.verify(mockApi).fetch();

inOrder.verify(mockApi).close();

}

}

**Output:**

****

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

**ExternalApi.java**

package com.example;

public class ExternalApi {

public void deleteData() {

}

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void removeData() {

api.deleteData();

}

}

**MyServiceExceptionTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

public class MyServiceExceptionTest {

@Test

public void testVoidMethodThrowsException() {

ExternalApi mockApi = mock(ExternalApi.class);

doThrow(new RuntimeException("Delete failed")).when(mockApi).deleteData();

MyService service = new MyService(mockApi);

assertThrows(RuntimeException.class, () -> {

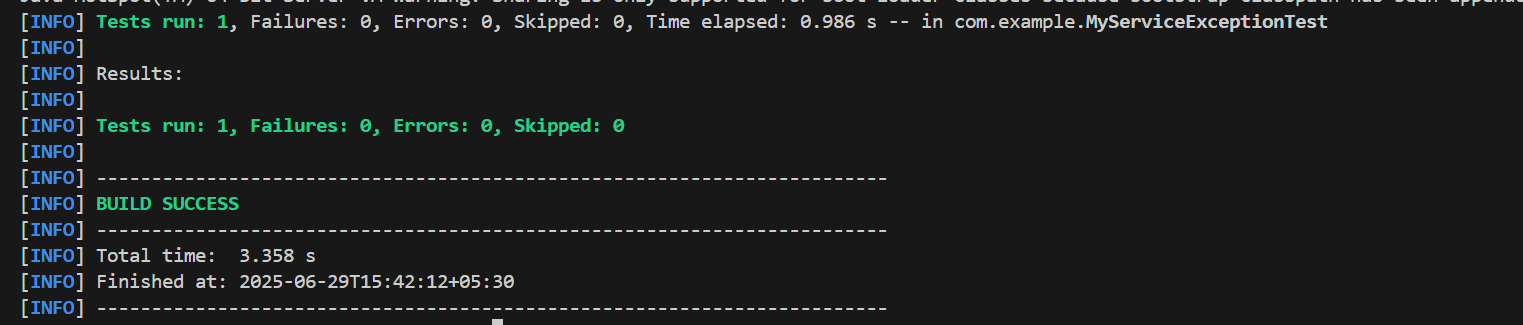
service.removeData();

});

verify(mockApi).deleteData();

}

}

**Output:**

**4.Advanced Mockito Hands-On Exercises**

**Exercise 1: Mocking Databases and Repositories**

**Repository.java**

package com.example;

public interface Repository {

String getData();

}

**Service.java**

package com.example;

public class Service {

private Repository repository;

public Service(Repository repository) {

this.repository = repository;

}

public String processData() {

return "Processed " + repository.getData();

}

}

**ServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

public class ServiceTest {

@Test

public void testServiceWithMockRepository() {

Repository mockRepository = mock(Repository.class);

when(mockRepository.getData()).thenReturn("Mock Data");

Service service = new Service(mockRepository);

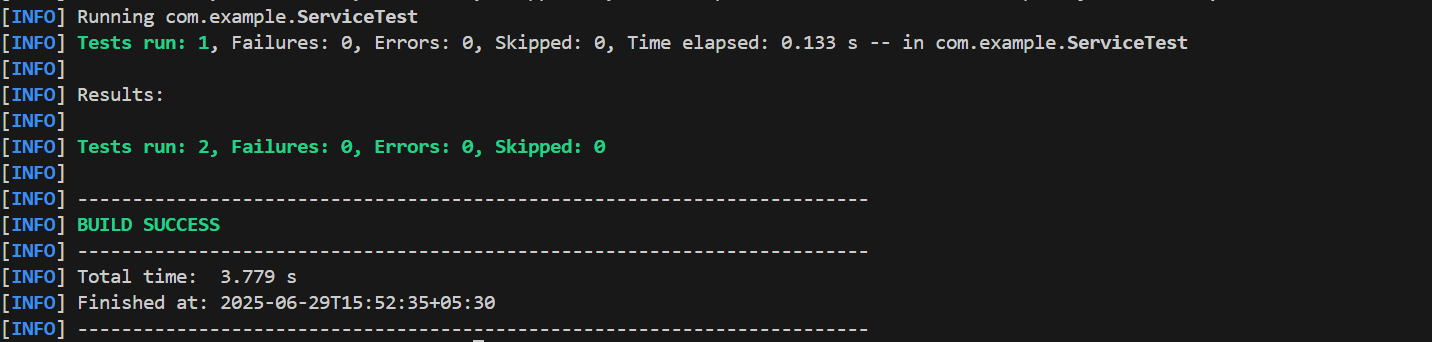
String result = service.processData();

assertEquals("Processed Mock Data", result);

}

}

**Output:**

****

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

**RestClient.java**

package com.example;

public interface RestClient {

String getResponse();

}

**ApiService.java**

package com.example;

public class ApiService {

private RestClient restClient;

public ApiService(RestClient restClient) {

this.restClient = restClient;

}

public String fetchData() {

return "Fetched " + restClient.getResponse();

}

}

**ApiServiceTest.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");

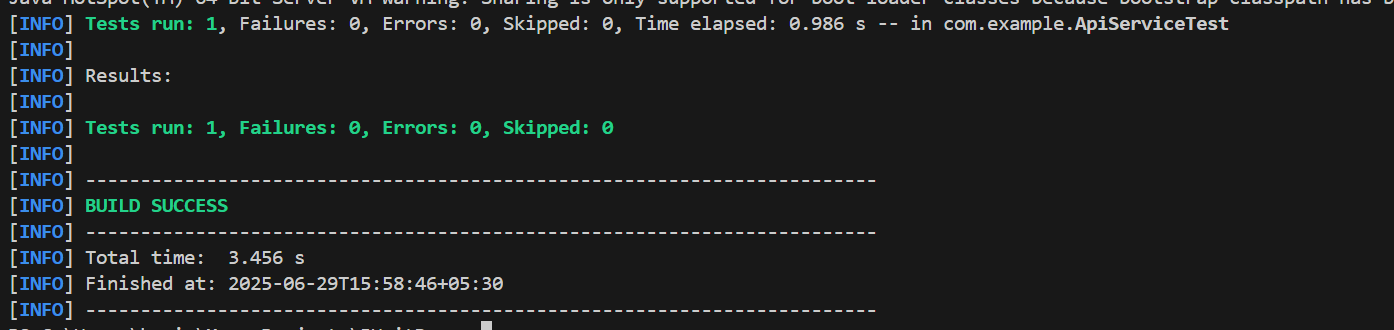
ApiService apiService = new ApiService(mockRestClient);

String result = apiService.fetchData();

assertEquals("Fetched Mock Response", result);

}

}

**Output:**

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

**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 FileReader fileReader;

private 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);

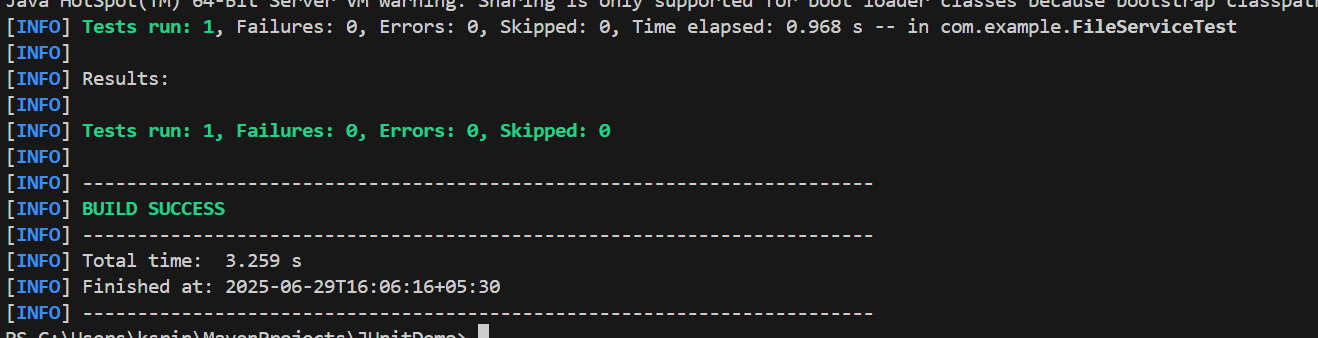
String result = fileService.processFile();

assertEquals("Processed Mock File Content", result);

verify(mockFileWriter).write("Processed Mock File Content");

}

}

**Output:**

**Exercise 4: Mocking Network Interactions**

**NetworkClient.java**

package com.example;

public interface NetworkClient {

String connect();

}

**NetworkService.java**

package com.example;

public class NetworkService {

private NetworkClient networkClient;

public NetworkService(NetworkClient networkClient) {

this.networkClient = networkClient;

}

public String connectToServer() {

String connection = networkClient.connect();

return "Connected to " + connection;

}

}

**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);

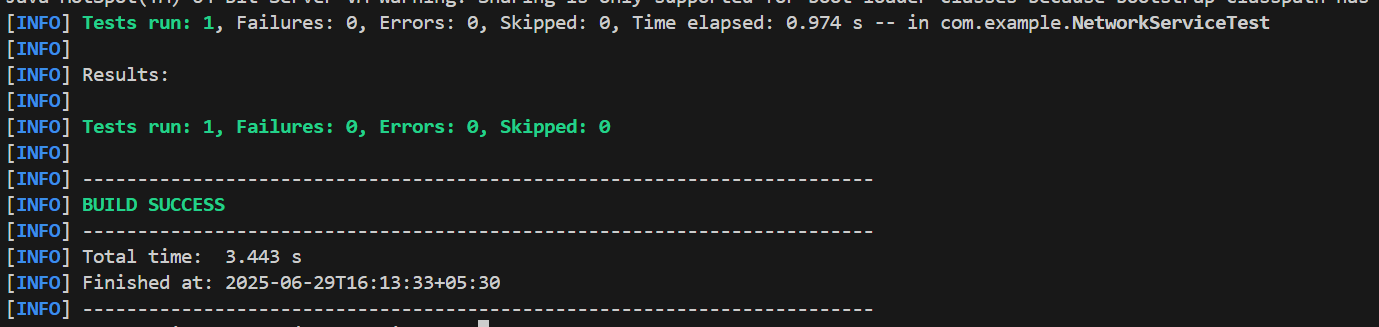
String result = networkService.connectToServer();

assertEquals("Connected to Mock Connection", result);

}

}

**Output:**



**Exercise 5: Mocking Multiple Return Values**

**Repository.java**

package com.example;

public interface Repository {

String getData();

}

**Service.java**

package com.example;

public class Service {

private 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();

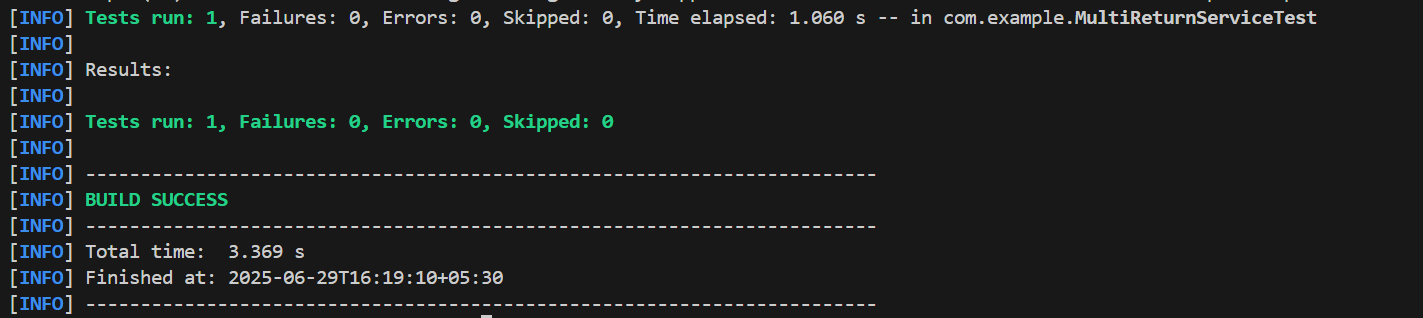
assertEquals("Processed First Mock Data", firstResult);

assertEquals("Processed Second Mock Data", secondResult);

}

}

**Output:**



**5.Spring Testing Exercises**

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

**CalculatorService.java**

package com.example;

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 calculator = new CalculatorService();

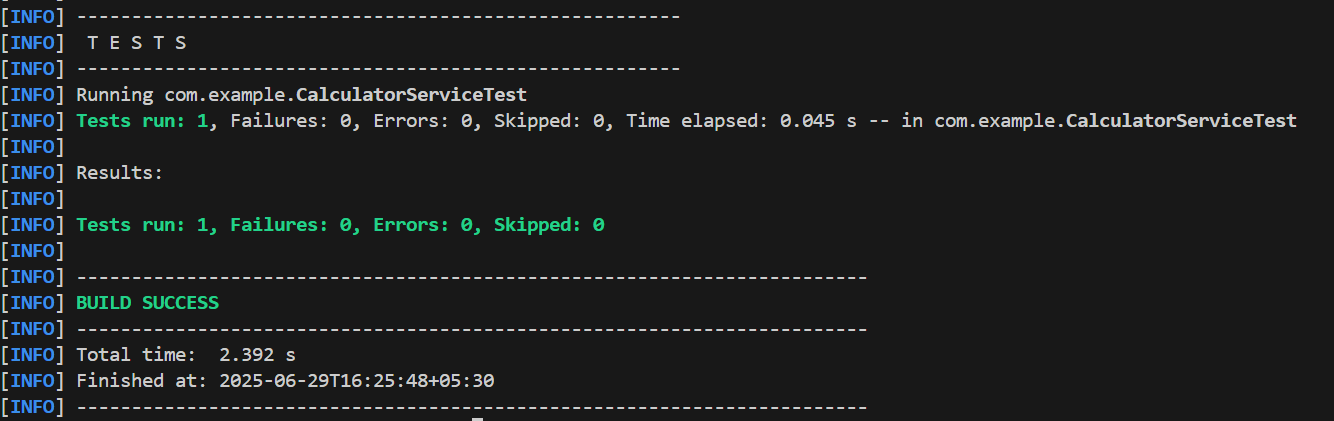
int result = calculator.add(10, 20);

assertEquals(30, result);

}

}

**Output:**



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

**User.java**

package com.example;

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 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;

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

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

package com.example;

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

import org.springframework.stereotype.Service;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public UserService(UserRepository userRepository) {

this.userRepository = userRepository;

}

public User getUserById(Long id) {

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

}

}

**UserServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

import java.util.Optional;

public class UserServiceTest {

@Test

public void testGetUserById() {

UserRepository mockRepo = mock(UserRepository.class);

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

when(mockRepo.findById(1L)).thenReturn(Optional.of(mockUser));

UserService userService = new UserService(mockRepo);

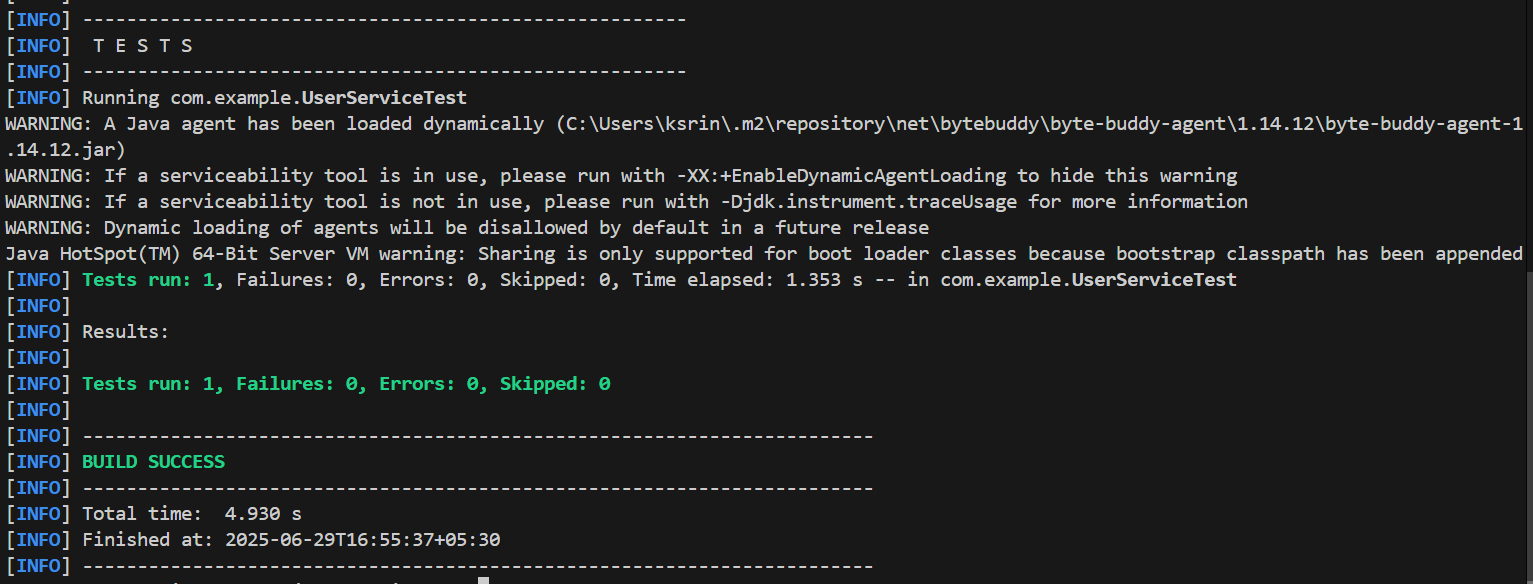
User result = userService.getUserById(1L);

assertNotNull(result);

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

}

}

**Output:**

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

**User.java**

package com.example;

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 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;

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

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

package com.example;

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);

}

}

**UserController.java**

package com.example;

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));

}

}

**UserControllerTest.java**

package com.example;

import static org.mockito.Mockito.\*;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

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

import com.fasterxml.jackson.databind.ObjectMapper;

import org.junit.jupiter.api.Test;

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

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

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

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

@WebMvcTest(UserController.class)

public class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

public void testGetUser() throws Exception {

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

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

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

.andExpect(status().isOk())

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

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

}

}

**JUnitDemoApplication.java**

package com.example;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class JUnitDemoApplication {

public static void main(String[] args) {

SpringApplication.run(JUnitDemoApplication.class, args);

}

}

**pom.xml**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>jakarta.persistence</groupId>

<artifactId>jakarta.persistence-api</artifactId>

<version>3.1.0</version>

</dependency>

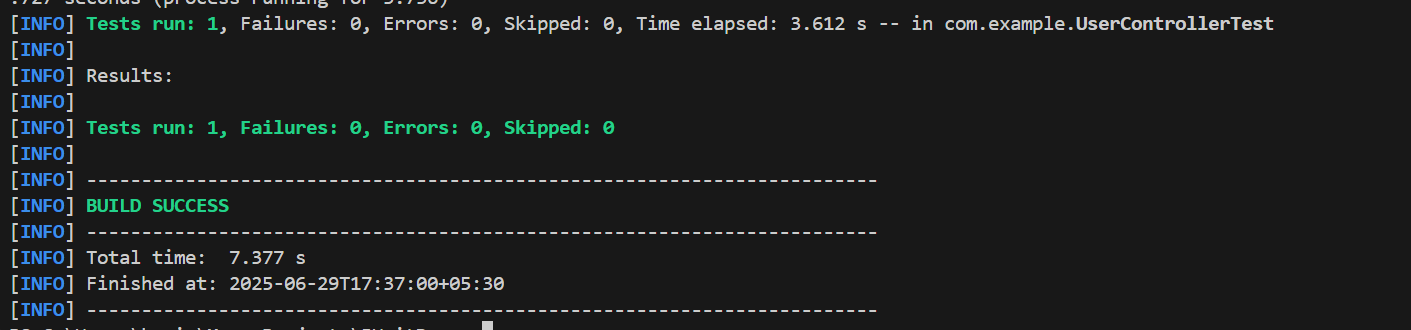
<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

**Output:**



**Exercise 4: Integration Test with Spring Boot**

**User.java**

package com.example;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class User {

@Id

private Long id;

private String name;

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;

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

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

package com.example;

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);

}

public User saveUser(User user) {

return userRepository.save(user);

}

}

**UserController.java**

package com.example;

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;

@PostMapping

public ResponseEntity<User> saveUser(@RequestBody User user) {

return ResponseEntity.ok(userService.saveUser(user));

}

@GetMapping("/{id}")

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

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

}

}

**UserIntegrationTest.java**

package com.example;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.junit.jupiter.api.Test;

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

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

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.http.MediaType;

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

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

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

@SpringBootTest

@AutoConfigureMockMvc

public class UserIntegrationTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private ObjectMapper objectMapper;

@Test

public void testCreateAndGetUser() throws Exception {

User user = new User();

user.setId(1L);

user.setName("Yashaswini");

mockMvc.perform(post("/users")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(user)))

.andExpect(status().isOk())

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

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

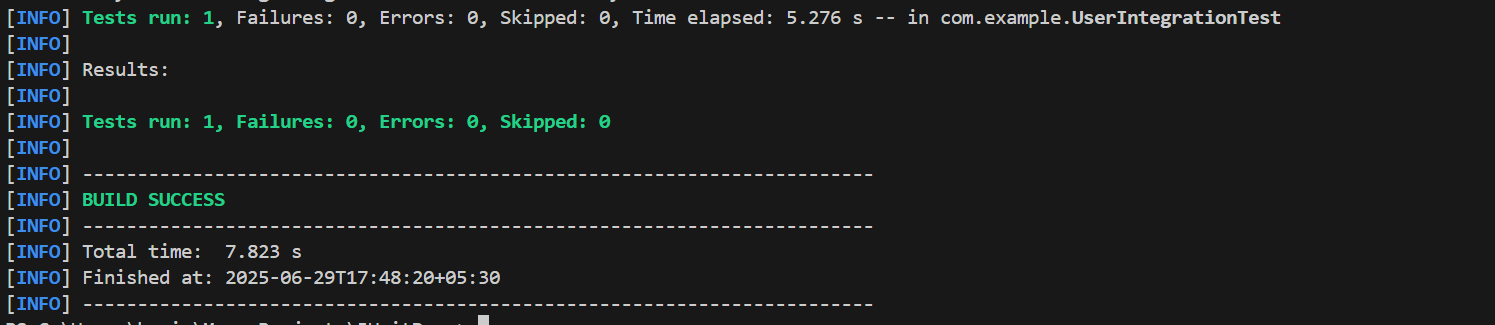
.andExpect(status().isOk())

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

}

}

**Output:**



**Exercise 5: Test Controller POST Endpoint**

**User.java**

package com.example;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "users")

public class User {

    @Id

    private Long id;

    private String name;

    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;

    }

}

**UserController.java**

package com.example;

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;

@PostMapping

public ResponseEntity<User> saveUser(@RequestBody User user) {

User saved = userService.saveUser(user);

return ResponseEntity.ok(saved);

}

}

**UserService.java**

package com.example;

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

import org.springframework.stereotype.Service;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public User saveUser(User user) {

return userRepository.save(user);

}

}

**UserRepository.java**

package com.example;

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

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserControllerTest.java**

package com.example;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;

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

import com.fasterxml.jackson.databind.ObjectMapper;

import org.junit.jupiter.api.Test;

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

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

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.http.MediaType;

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

@SpringBootTest

@AutoConfigureMockMvc

public class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private ObjectMapper objectMapper;

@Test

public void testCreateUser() throws Exception {

User user = new User();

user.setId(1L);

user.setName("John");

mockMvc.perform(post("/users")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(user)))

.andExpect(status().isOk())

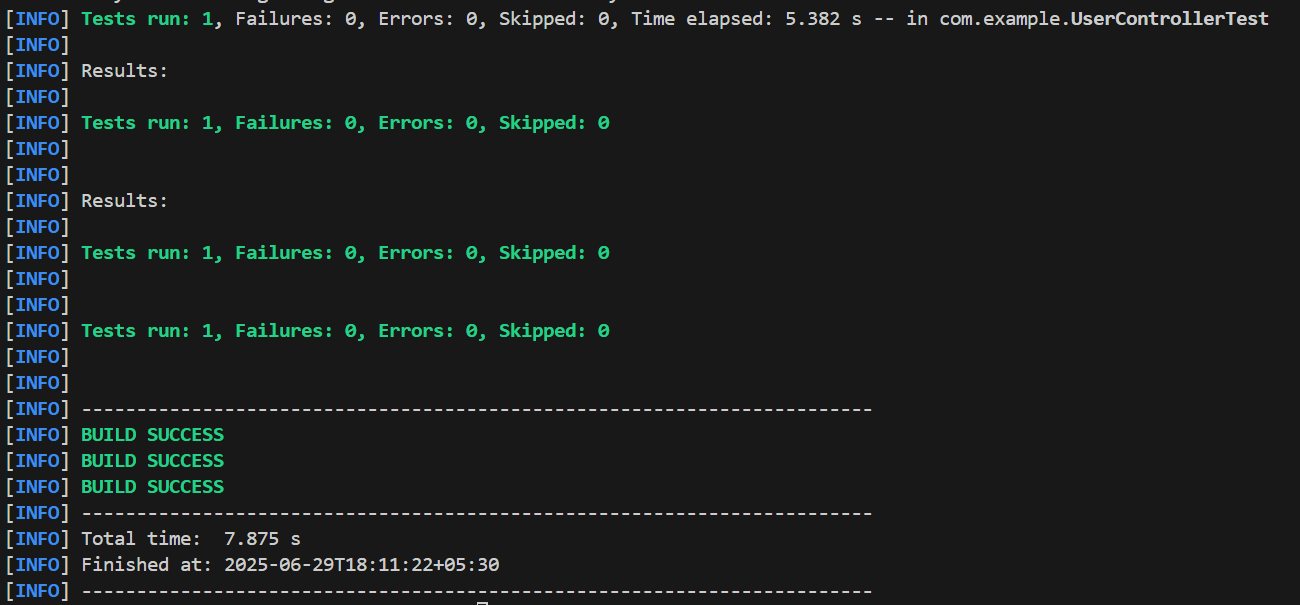
.andExpect(jsonPath("$.id").value(1))

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

}

}

**Output:**



**Exercise 6: Test Service Exception Handling**

**UserNotFoundException.java**

package com.example;

public class UserNotFoundException extends RuntimeException {

public UserNotFoundException(String message) {

super(message);

}

}

**UserService.java**

package com.example;

import org.springframework.stereotype.Service;

import java.util.Optional;

@Service

public class UserService {

private final UserRepository userRepository;

public UserService(UserRepository userRepository) {

this.userRepository = userRepository;

}

public User getUserById(Long id) {

return userRepository.findById(id)

.orElseThrow(() -> new UserNotFoundException("User not found with id " + id));

}

}

**UserServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import java.util.Optional;

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

import static org.mockito.Mockito.\*;

public class UserServiceTest {

@Test

void testGetUserById\_UserNotFound() {

UserRepository mockRepo = Mockito.mock(UserRepository.class);

UserService service = new UserService(mockRepo);

Long testId = 1L;

when(mockRepo.findById(testId)).thenReturn(Optional.empty());

UserNotFoundException thrown = assertThrows(UserNotFoundException.class, () -> {

service.getUserById(testId);

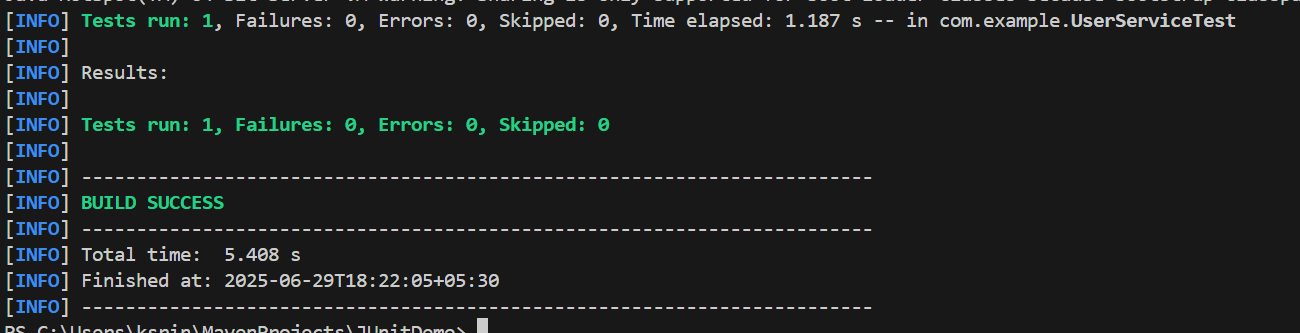
});

assertEquals("User not found with id 1", thrown.getMessage());

}

}

**Output:**



**Exercise 7: Test Custom Repository Query**

**User.java**

package com.example;

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 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;

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

import java.util.List;

public interface UserRepository extends JpaRepository<User, Long> {

List<User> findByName(String name);

}

**UserRepositoryTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;

import java.util.List;

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

@DataJpaTest

public class UserRepositoryTest {

@Autowired

private UserRepository userRepository;

@Test

public void testFindByName() {

User user1 = new User(1L, "Alice");

User user2 = new User(2L, "Alice");

User user3 = new User(3L, "Bob");

userRepository.save(user1);

userRepository.save(user2);

userRepository.save(user3);

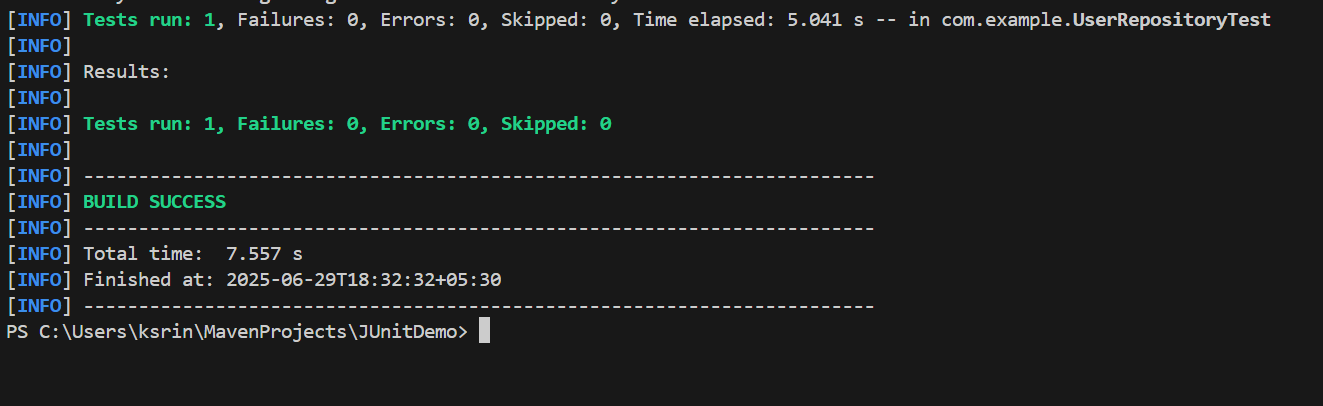
List<User> result = userRepository.findByName("Alice");

assertEquals(2, result.size());

assertTrue(result.stream().allMatch(user -> "Alice".equals(user.getName())));

}

}

**Output:**

**Exercise 8: Test Controller Exception Handling**

**User.java**

package com.example;

public class User {

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

public String getName() { return name; }

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

}

**UserService.java**

package com.example;

import java.util.NoSuchElementException;

public interface UserService {

User getUserById(Long id) throws NoSuchElementException;

}

**UserController.java**

package com.example;

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> getUserById(@PathVariable Long id) {

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

}

}

**GlobalExceptionHandler.java**

package com.example;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.NoSuchElementException;

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(NoSuchElementException.class)

public ResponseEntity<String> handleNotFound(NoSuchElementException ex) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("User not found");

}

}

**UserControllerExceptionTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

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

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

import org.springframework.http.MediaType;

import org.springframework.test.context.ContextConfiguration;

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

import java.util.NoSuchElementException;

import static org.mockito.Mockito.when;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

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

@WebMvcTest(UserController.class)

@ContextConfiguration(classes = {UserController.class, GlobalExceptionHandler.class})

public class UserControllerExceptionTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

public void testUserNotFoundException() throws Exception {

when(userService.getUserById(999L)).thenThrow(new NoSuchElementException("User not found"));

mockMvc.perform(get("/users/999")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isNotFound())

.andExpect(content().string("User not found"));

}

}

**JUnitDemoApplication.java**

package com.example;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class JUnitDemoApplication {

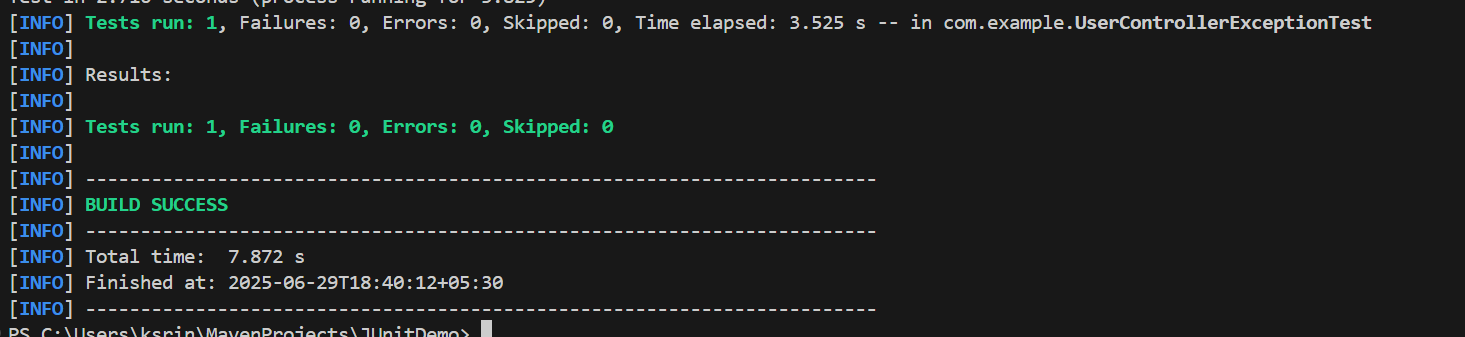
public static void main(String[] args) {

SpringApplication.run(JUnitDemoApplication.class, args);

}

}

**Output:**



**Exercise 9: Parameterized Test with Junit**

**UserUtil.java**

package com.example;

public class UserUtil {

public static boolean isValidName(String name) {

return name != null && !name.trim().isEmpty();

}

}

**UserUtilTest.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 UserUtilTest {

@ParameterizedTest

@ValueSource(strings = {"Alice", "Bob", "Charlie"})

void testValidNames(String name) {

assertTrue(UserUtil.isValidName(name), "Name should be valid");

}

@ParameterizedTest

@ValueSource(strings = {"", " ", "\t", "\n"})

void testInvalidNames(String name) {

assertFalse(UserUtil.isValidName(name), "Name should be invalid");

}

@ParameterizedTest

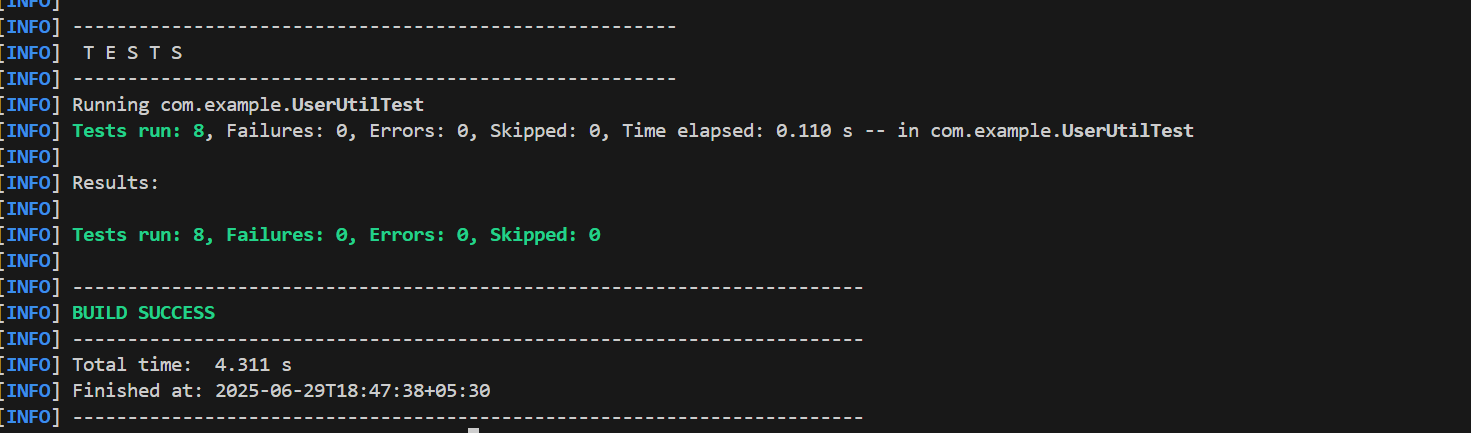
@ValueSource(strings = {""})

void testEdgeCaseEmptyString(String name) {

assertFalse(UserUtil.isValidName(name), "Empty string is not valid");

}

}

**Output:**

**6.Mocking Dependencies in Spring Tests using Mockito**

**Exercise 1: Mocking a Service Dependency in a Controller Test**

**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

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>spring-mock-demo</artifactId>

<version>1.0-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.2.5</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**DemoApplication.java**

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

**User.java**

package com.example.demo;

import javax.persistence.Entity;

import javax.persistence.Id;

@Entity

public class User {

@Id

private Long id;

private String name;

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;

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

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

package com.example.demo;

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);

}

}

**UserController.java**

package com.example.demo;

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));

}

}

**UserControllerTest.java**

package com.example.demo;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.when;

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 org.springframework.http.MediaType;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

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

@WebMvcTest(UserController.class)

public class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

public void testGetUserById\_ReturnsUser() throws Exception {

User mockUser = new User();

mockUser.setId(1L);

mockUser.setName("Yashaswini");

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

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

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

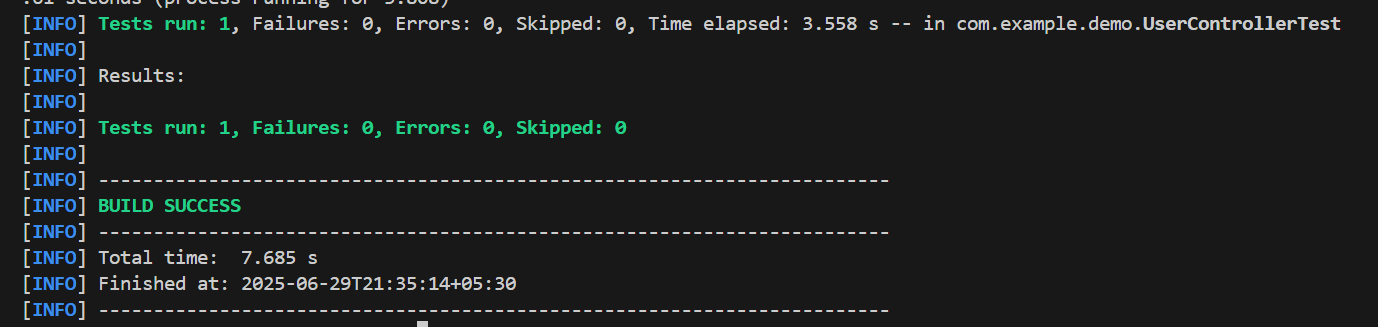
.andExpect(jsonPath("$.id").value(1))

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

}

}

**Output:**



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

**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

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>spring-mock-demo</artifactId>

<version>1.0-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.1.5</version>

<relativePath/>

</parent>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<version>3.1.5</version>

</plugin>

</plugins>

</build>

</project>

**DemoApplication.java**

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

**User.java**

package com.example.demo;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class User {

@Id

private Long id;

private String name;

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;

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

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

package com.example.demo;

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;

import org.junit.jupiter.api.BeforeEach;

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.junit.jupiter.api.Assertions.\*;

import static org.mockito.Mockito.\*;

public class UserServiceTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

@BeforeEach

void setUp() {

MockitoAnnotations.openMocks(this);

}

@Test

public void testGetUserById\_ReturnsUser() {

User mockUser = new User();

mockUser.setId(1L);

mockUser.setName("Yashaswini");

when(userRepository.findById(1L)).thenReturn(Optional.of(mockUser));

User result = userService.getUserById(1L);

assertNotNull(result);

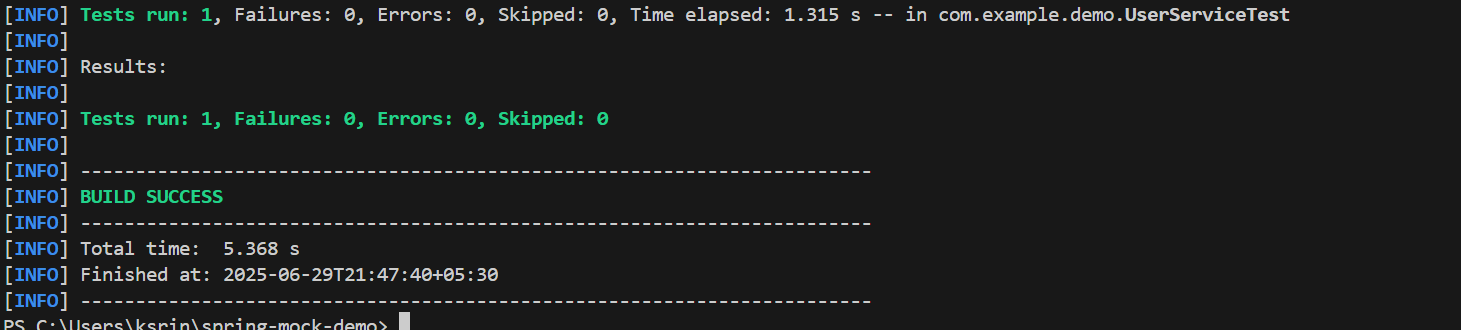
assertEquals("Yashaswini", result.getName());

assertEquals(1L, result.getId());

}

}

**Output:**



**Exercise 3: Mocking a Service Dependency in an Integration Test**

**DemoApplication.java**

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

**User.java**

package com.example.demo;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class User {

@Id

private Long id;

private String name;

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;

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

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

package com.example.demo;

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);

}

}

**UserController.java**

package com.example.demo;

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));

}

}

**TestSecurityConfig.java**

package com.example.demo;

import org.springframework.boot.test.context.TestConfiguration;

import org.springframework.context.annotation.Bean;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.web.SecurityFilterChain;

@TestConfiguration

public class TestSecurityConfig {

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeHttpRequests(auth -> auth.anyRequest().permitAll());

return http.build();

}

}

**UserIntegrationTest.java**

package com.example.demo;

import org.junit.jupiter.api.Test;

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

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

import org.springframework.boot.test.context.SpringBootTest;

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

import org.springframework.context.annotation.Import;

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

import static org.mockito.Mockito.when;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

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

@SpringBootTest(classes = DemoApplication.class)

@AutoConfigureMockMvc

@Import(TestSecurityConfig.class)

public class UserIntegrationTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

void testGetUserById() throws Exception {

User mockUser = new User();

mockUser.setId(1L);

mockUser.setName("Yashaswini");

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

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

.andExpect(status().isOk())

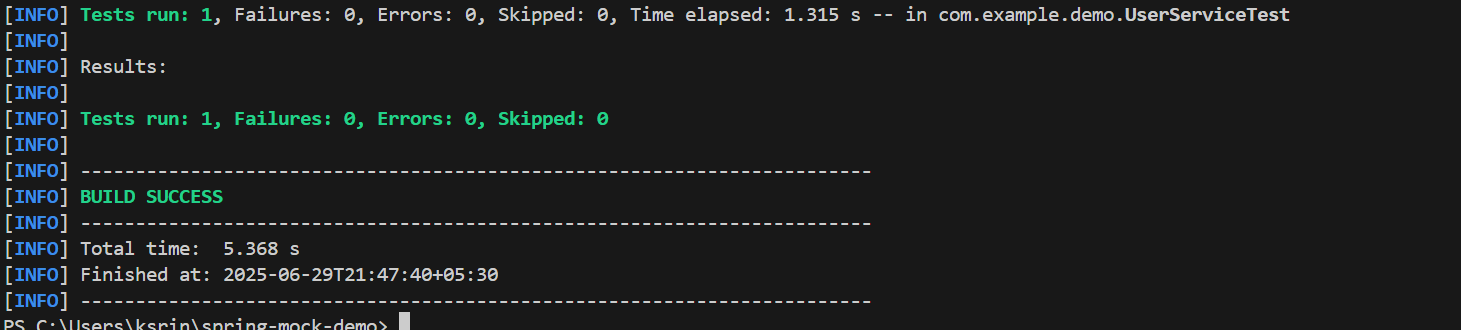
.andExpect(jsonPath("$.id").value(1))

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

}

}

**Output:**



**7.Logging using SLF4J**

**Exercise 1: Logging Error Messages and Warning Levels**

**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.example.demo</groupId>

<artifactId>logging-example</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.0.0</version>

<configuration>

<mainClass>com.example.demo.LoggingExample</mainClass>

</configuration>

</plugin>

</plugins>

</build>

</project>

**LoggingExample.java**

package com.example.demo;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);

public static void main(String[] args) {

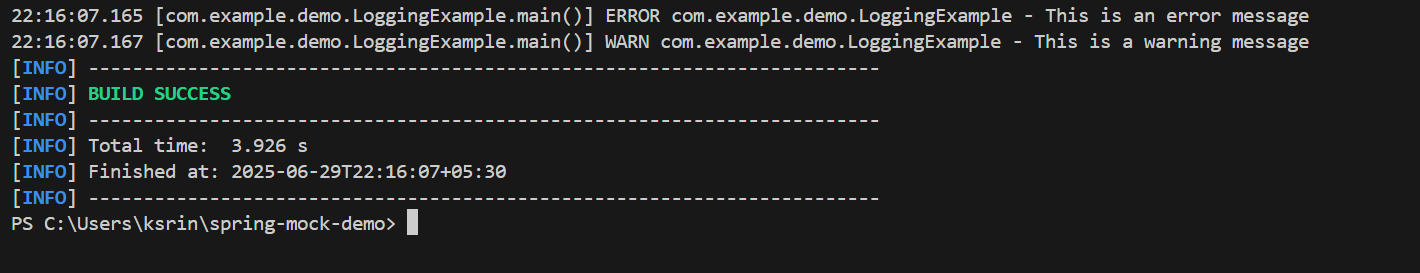
logger.error("This is an error message");

logger.warn("This is a warning message");

}

}

Output:



**Exercise 2: Parameterized Logging**

**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.example.demo</groupId>

<artifactId>parameterized-logging-example</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.0.0</version>

<configuration>

<mainClass>com.example.demo.ParameterizedLoggingExample</mainClass>

</configuration>

</plugin>

</plugins>

</build>

</project>

**ParameterizedLoggingExample.java**

package com.example.demo;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class ParameterizedLoggingExample {

private static final Logger logger = LoggerFactory.getLogger(ParameterizedLoggingExample.class);

public static void main(String[] args) {

String user = "Yashaswini";

int orderId = 12345;

logger.info("User {} placed order with ID {}", user, orderId);

int retryCount = 3;

logger.warn("Retry attempt {} for order ID {}", retryCount, orderId);

try {

int result = 10 / 0; // will throw ArithmeticException

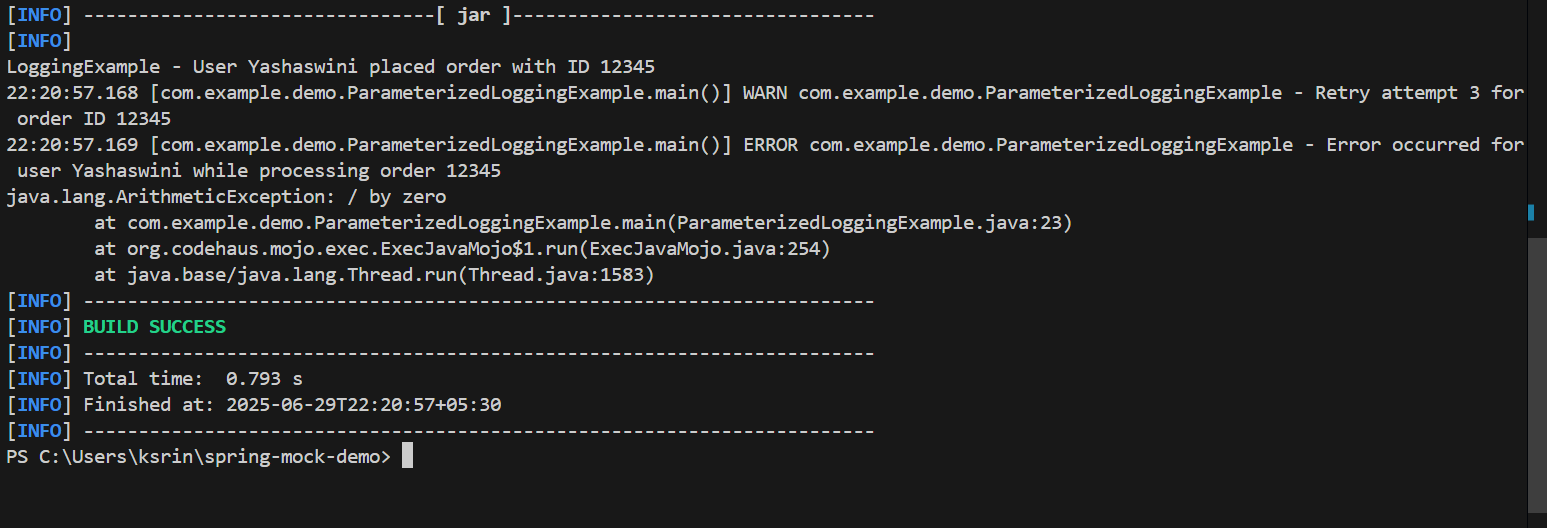
} catch (ArithmeticException e) {

logger.error("Error occurred for user {} while processing order {}", user, orderId, e);

}

}

}

**Output:**

**Exercise 3: Using Different Appenders**

**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.example.demo</groupId>

<artifactId>logging-appenders-example</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.0.0</version>

<configuration>

<mainClass>com.example.demo.DifferentAppendersExample</mainClass>

</configuration>

</plugin>

</plugins>

</build>

</project>

**DifferentAppendersExample.java**

package com.example.demo;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class DifferentAppendersExample {

private static final Logger logger = LoggerFactory.getLogger(DifferentAppendersExample.class);

public static void main(String[] args) {

logger.debug("Debug message: Application started");

logger.info("Info message: Running the application");

logger.warn("Warning message: Low disk space");

logger.error("Error message: Exception occurred", new RuntimeException("Test exception"));

}

}

**logback.xml**

<configuration>

<appender name="console" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<appender name="file" class="ch.qos.logback.core.FileAppender">

<file>app.log</file>

<encoder>

<pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<root level="debug">

<appender-ref ref="console" />

<appender-ref ref="file" />

</root>

</configuration>

**Output:**