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

[**MyServiceTest.ja**](http://myservicetest.ja)**va** verifies that **MyService.java** correctly fetches data from a mocked **ExternalApi.java** using stubbing.

**ExternalApi.java:**

package com.example;

public interface ExternalApi {

String getData();

void reset();

}

**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();

}

public void resetApi() {

api.reset();

}

}

[**MyServiceTest.ja**](http://myservicetest.ja)**va:**

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 mapi = mock(ExternalApi.class);

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

MyService s = new MyService(mapi);

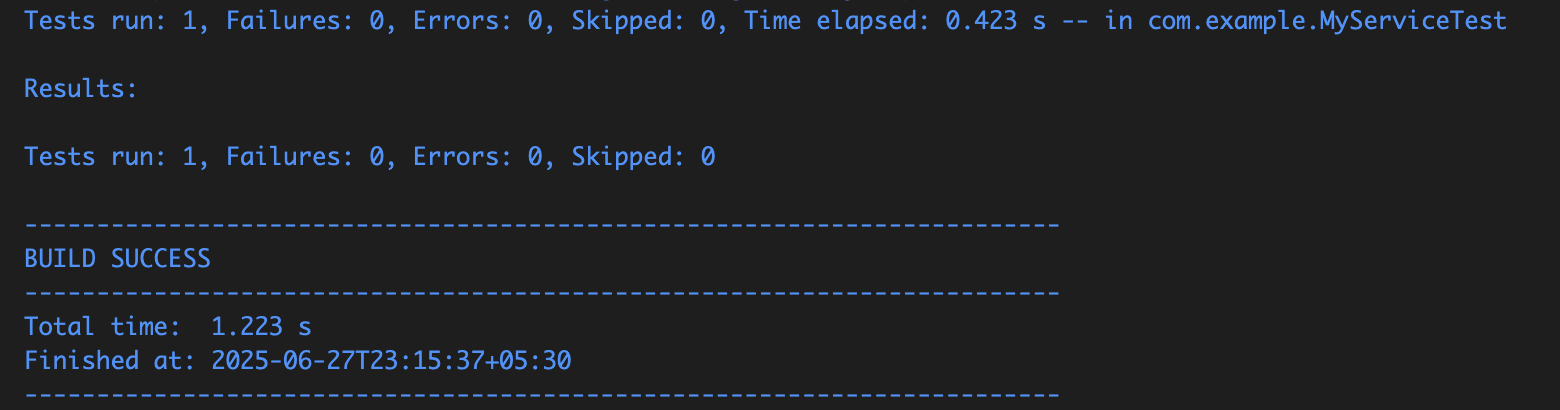
String res = s.fetchData();

assertEquals("Mock Data", res);

}

}

**Output:**

****

**Exercise 2: Verifying Interactions (MANDATORY)**

Verifies that the **getData()** method of the mocked **ExternalApi** is called when **MyService.fetchData()** is executed.

**InteractionTest.java:**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.verify;

public class InteractionTest {

@Test

public void testVerifyInteraction() {

ExternalApi mapi = mock(ExternalApi.class);

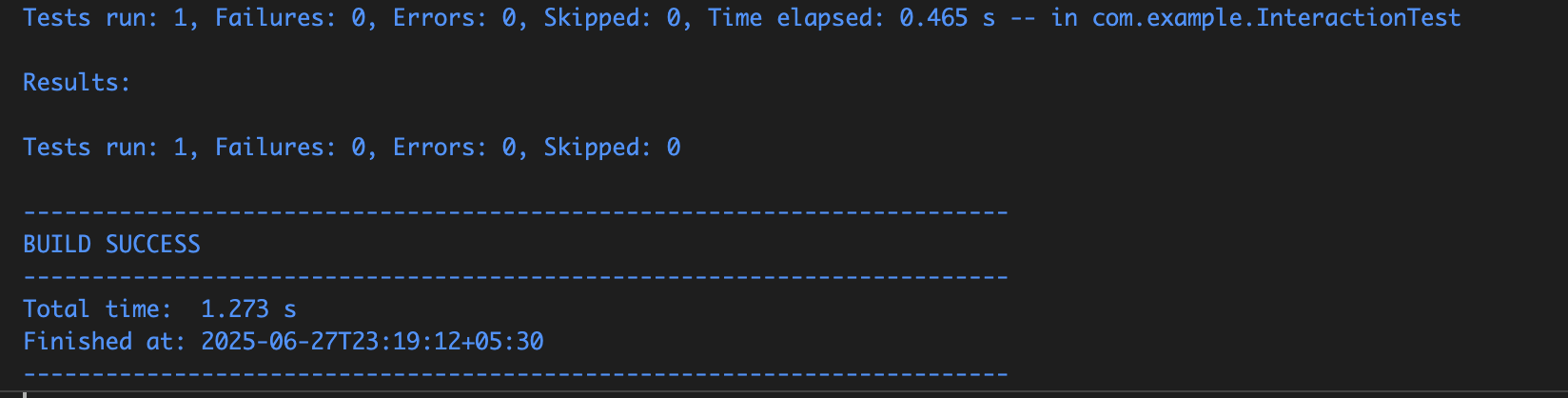
MyService s = new MyService(mapi);

s.fetchData();

verify(mapi).getData();

}

}

**Output:**   
  


**Exercise 3: Argument Matching**

**ArgumentMatchingTest.java** ensures the method is called with a specific argument using argument matchers.

**ArgumentMatchingTest.java:**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.ArgumentMatchers.eq;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.verify;

public class ArgumentMatchingTest {

interface Calculator {

int add(int a, int b);

}

@Test

public void testAddMethodWithArguments() {

Calculator mcal = mock(Calculator.class);

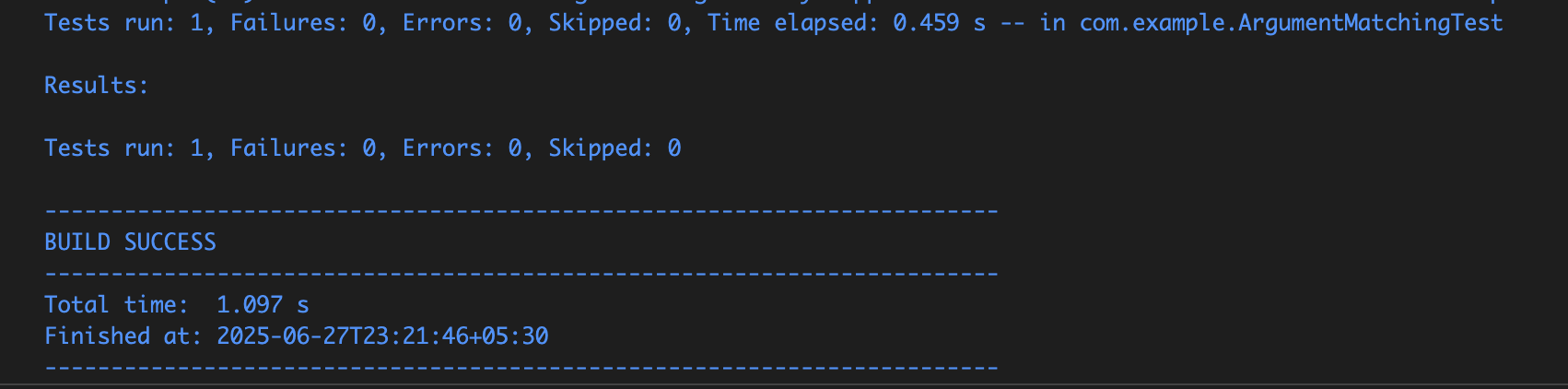
mcal.add(5, 10);

verify(mcal).add(eq(5), eq(10));

}

}

**Output:**

****

**Exercise 4: Handling Void Methods**

**VoidMethodTest.java** mocks a void method and verifies it was invoked without throwing exceptions.

**VoidMethodTest.java:**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.ArgumentMatchers.anyString;

import static org.mockito.Mockito.doNothing;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.verify;

interface Notifier {

void send(String message);

}

public class VoidMethodTest {

@Test

public void testVoidMethodInteraction() {

Notifier mn = mock(Notifier.class);

doNothing().when(mn).send(anyString());

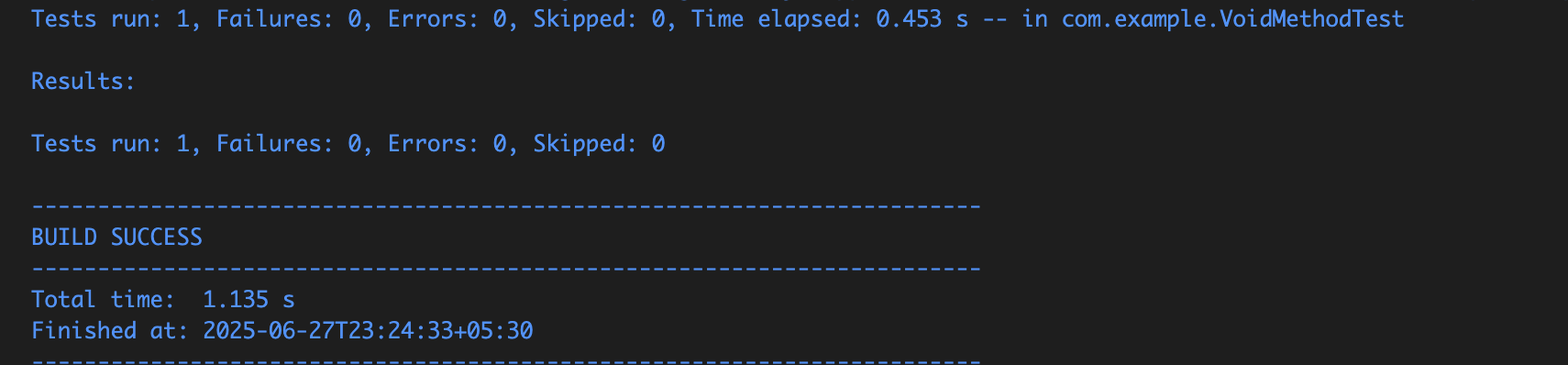
mn.send("Hello, Mockito!");

verify(mn).send("Hello, Mockito!");

}

}

**Output:**



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

**MultipleReturnsTest.java t**ests a method returning different outputs across multiple calls.

**MultipleReturnsTest.java**:

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.times;

import static org.mockito.Mockito.verify;

import static org.mockito.Mockito.when;

public class MultipleReturnsTest {

interface DataProvider {

String fetch();

}

@Test

public void testMultipleReturnValues() {

DataProvider mp = mock(DataProvider.class);

when(mockProvider.fetch()).thenReturn("First Call") .thenReturn("Second Call") .thenReturn("Third Call");

System.out.println(mp.fetch());

System.out.println(mp.fetch());

System.out.println(mp.fetch());

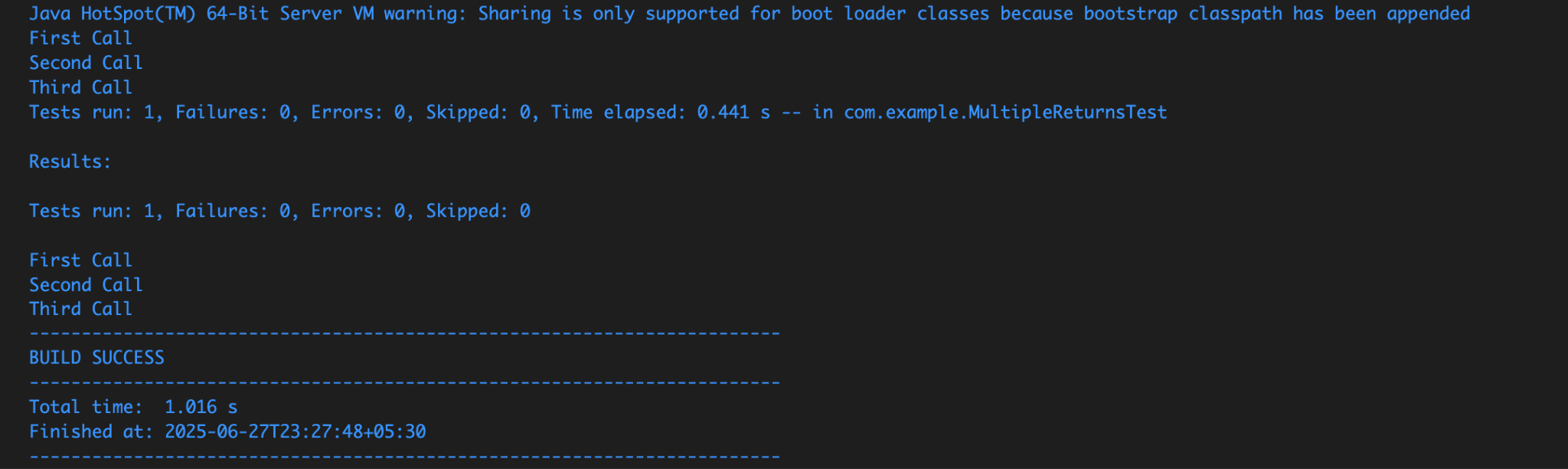
verify(mp, times(3)).fetch();

}

}

**Output:**

Prints First Call, Second Call, Third Call

****

**Exercise 6: Verifying Interaction Order**

**InteractionOrderTest.java** checks that multiple method calls on a mock occur in the correct sequence.

**InteractionOrderTest.java:**

package com.example;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

import static org.mockito.Mockito.inOrder;

import static org.mockito.Mockito.mock;

public class InteractionOrderTest {

interface Service {

void stepOne();

void stepTwo();

void stepThree();

}

@Test

public void testMethodCallOrder() {

Service ms = mock(Service.class);

ms.stepOne();

ms.stepTwo();

ms.stepThree();

InOrder io = inOrder(ms);

io.verify(ms).stepOne();

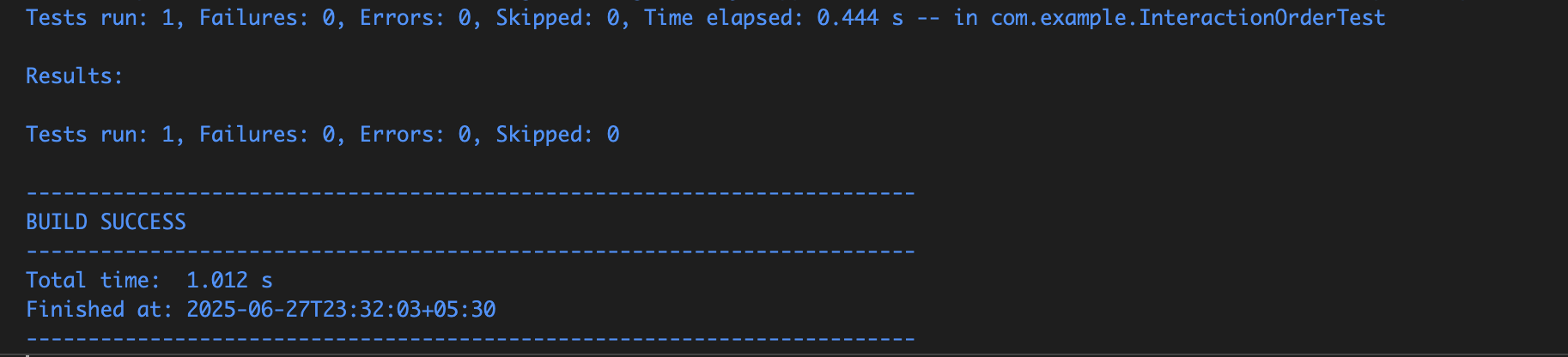
io.verify(ms).stepTwo();

io.verify(ms).stepThree();

}

}

**Output:**



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

**VoidMethodExceptionTest.java** stubs a void method to throw an exception and verifies the exception is handled.

**VoidMethodExceptionTest.java:**

package com.example;

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

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

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.doThrow;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.verify;

public class VoidMethodExceptionTest {

@Test

public void testVoidMethodThrowsException() {

ExternalApi mapi = mock(ExternalApi.class);

doThrow(new RuntimeException("API failure")).when(mapi).reset();

MyService s = new MyService(mapi);

Exception e = assertThrows(RuntimeException.class, () -> {

s.resetApi();

});

assertEquals("API failure", e.getMessage());

verify(mapi).reset();

}

}

**Output:**  
