Mockito Hands-On Exercises

# Exercise 1: Mocking and Stubbing

Scenario:

You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

Steps:

1. Create a mock object for the external API.
2. Stub the methods to return predefined values.
3. Write a test case that uses the mock object.

Solution Code:

package com.example.junit\_demo;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

public class MyServiceTest {

*@Test*

void testExternalApi() {

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

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

MyService service = new MyService(mockApi);

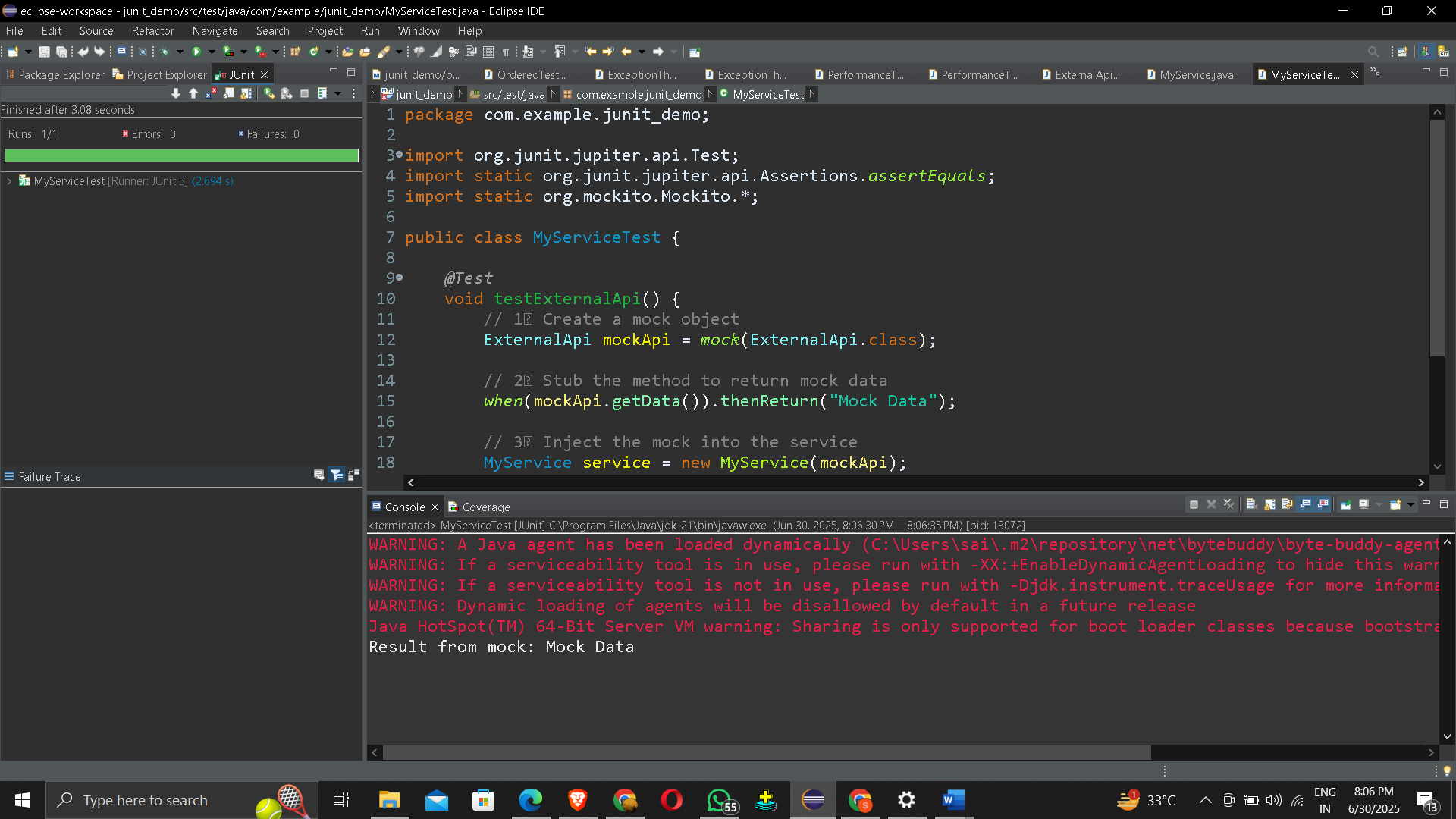
String result = service.fetchData();

System.***out***.println("Result from mock: " + result);

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

}

}



# Exercise 2: Verifying Interactions

Scenario:

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

Steps:

1. Create a mock object.
2. Call the method with specific arguments.
3. Verify the interaction.

Solution Code:

package com.example.junit\_demo;

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);

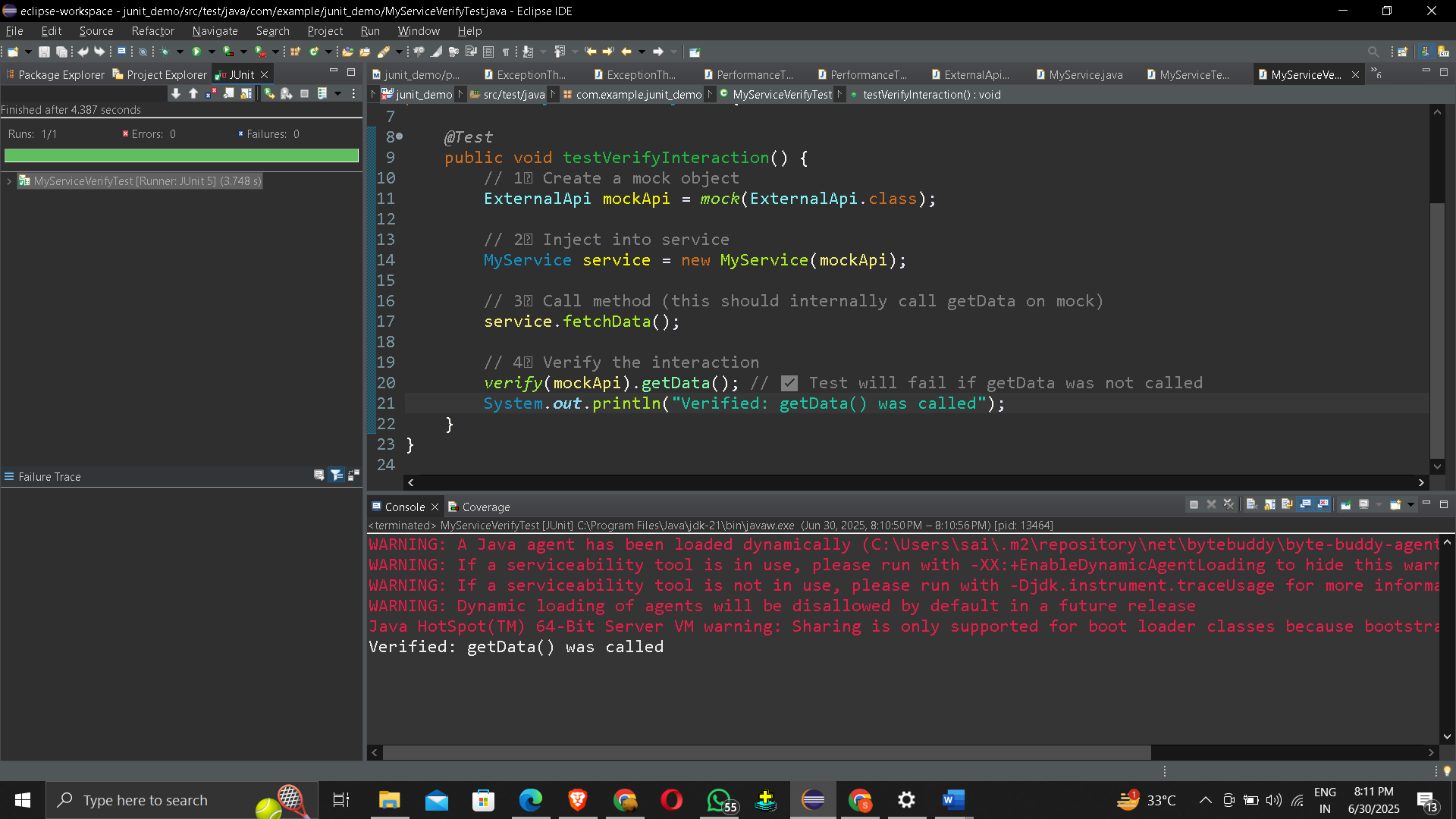
service.fetchData();

verify(mockApi).getData(); // Test will fail if getData was not called

System.out.println("Verified: getData() was called");

}

}



# Exercise 3: Argument Matching

Scenario:

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

Steps:

1. Create a mock object.
2. Call the method with specific arguments.
3. Use argument matchers to verify the interaction.

package com.example.junit\_demo;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

public class MyServiceArgumentTest {

*@Test*

void testArgumentMatching() {

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

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

MyService service = new MyService(mockApi);

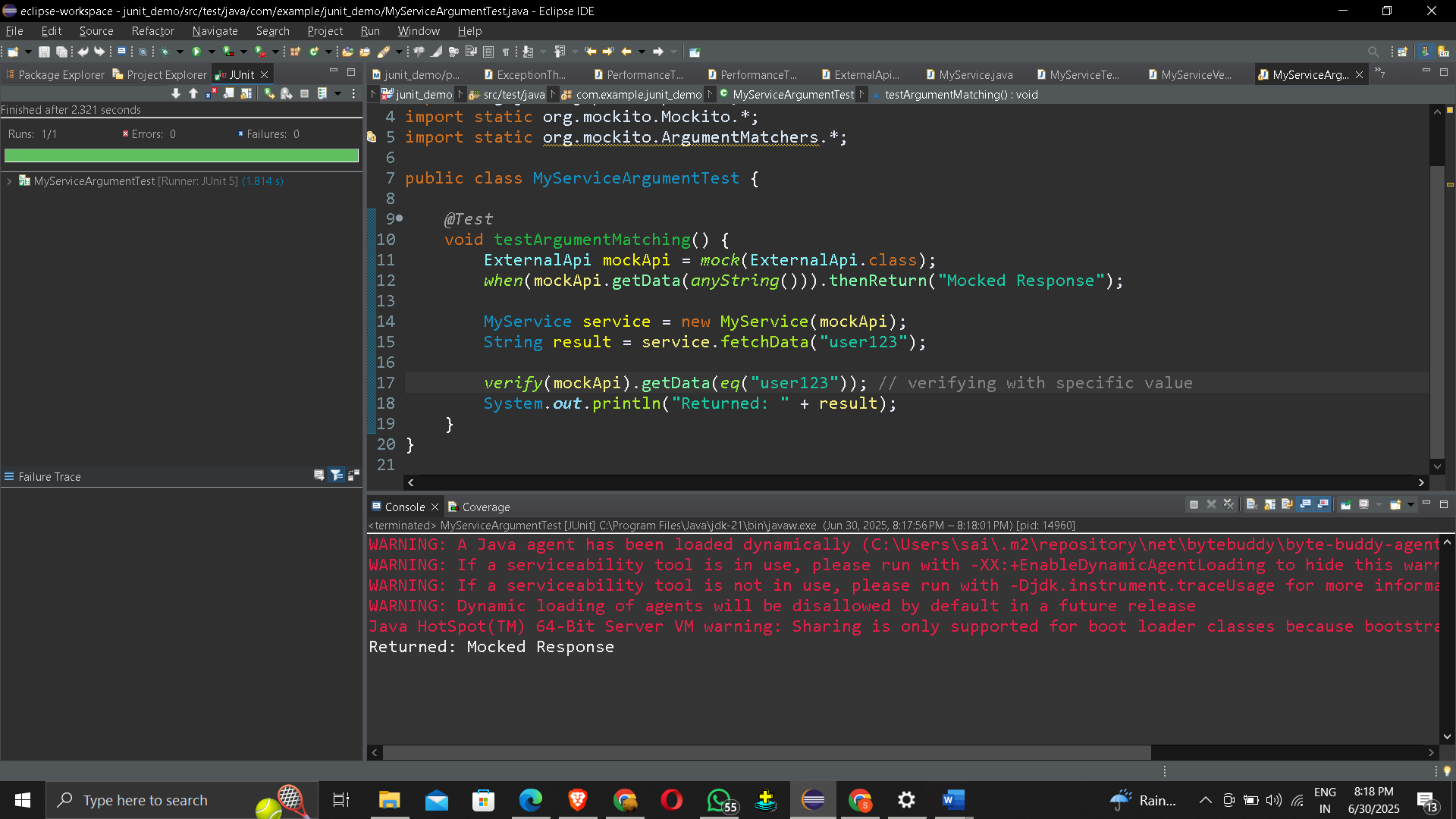
String result = service.fetchData("user123");

*verify*(mockApi).getData(*eq*("user123")); // verifying with specific value

System.***out***.println("Returned: " + result);

}

}



# Exercise 4: Handling Void Methods

Scenario:

You need to test a void method that performs some action.

Steps:

1. Create a mock object.
2. Stub the void method.
3. Verify the interaction.

package com.example.junit\_demo;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class NotificationServiceTest {

@Test

void testVoidMethod\_doNothing() {

ExternalApi mockApi = mock(ExternalApi.class);

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

NotificationService service = new NotificationService(mockApi);

service.notifyUser("Hello!");

verify(mockApi).sendNotification("Hello!");

}

@Test

void testVoidMethod\_doThrow() {

ExternalApi mockApi = mock(ExternalApi.class);

doThrow(new RuntimeException("Failed to send")).when(mockApi).sendNotification("fail");

NotificationService service = new NotificationService(mockApi);

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

service.notifyUser("fail");

});

assertEquals("Failed to send", exception.getMessage());

verify(mockApi).sendNotification("fail");

}

@Test

void testVoidMethod\_doAnswer() {

ExternalApi mockApi = mock(ExternalApi.class);

doAnswer(invocation -> {

String msg = invocation.getArgument(0);

System.out.println("Intercepted message: " + msg);

return null;

}).when(mockApi).sendNotification(anyString());

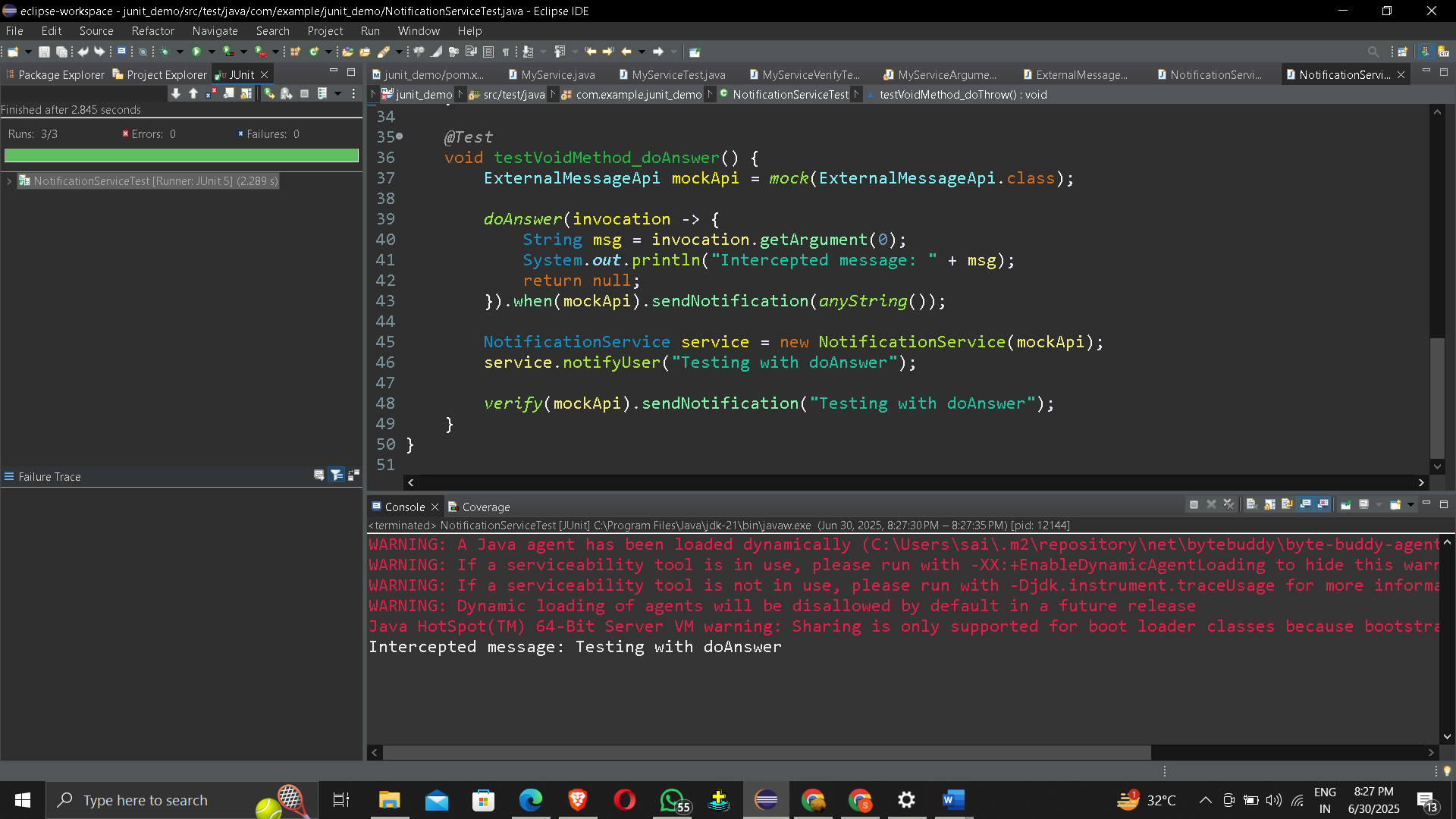
NotificationService service = new NotificationService(mockApi);

service.notifyUser("Testing with doAnswer");

verify(mockApi).sendNotification("Testing with doAnswer");

}

}



# Exercise 5: Mocking and Stubbing with Multiple Returns

Scenario:

You need to test a service that depends on an external API with multiple return values.

Steps:

1. Create a mock object for the external API.
2. Stub the methods to return different values on consecutive calls.
3. Write a test case that uses the mock object.

package com.example.junit\_demo;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class OrderServiceTest {

@Test

void testOrderStatusWithMultipleReturns() {

OrderApi mockApi = mock(OrderApi.class);

when(mockApi.getOrderStatus())

.thenReturn("Placed")

.thenReturn("Packed")

.thenReturn("Shipped")

.thenReturn("Delivered");

OrderService service = new OrderService(mockApi);

assertEquals("Placed", service.trackOrder());

assertEquals("Packed", service.trackOrder());

assertEquals("Shipped", service.trackOrder());

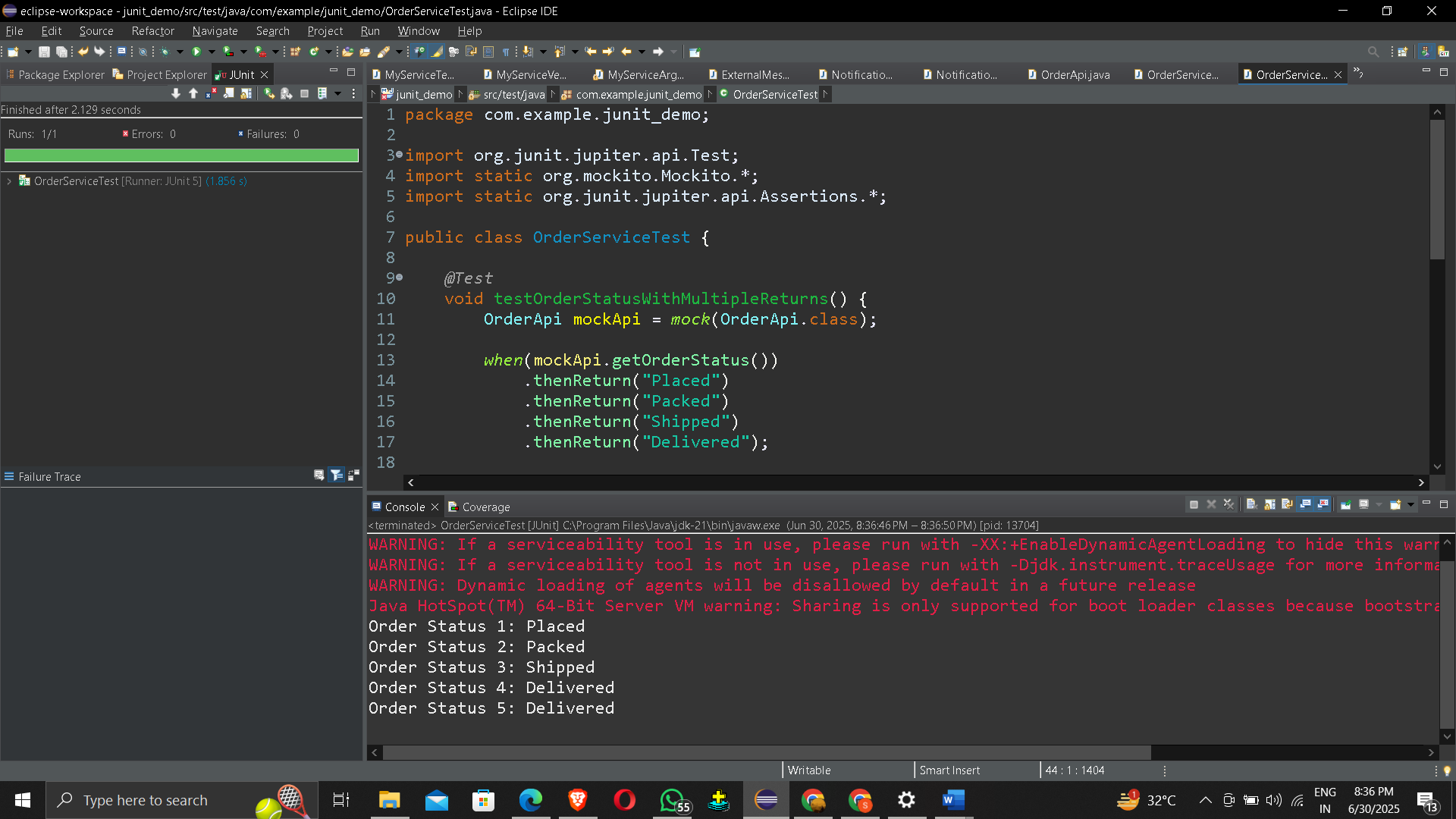
assertEquals("Delivered", service.trackOrder());

assertEquals("Delivered", service.trackOrder()); // last return value repeats

verify(mockApi, times(5)).getOrderStatus();

}

}



# Exercise 6: Verifying Interaction Order

Scenario:

You need to ensure that methods are called in a specific order.

Steps:

1. Create a mock object.
2. Call the methods in a specific order.
3. Verify the interaction order.

package com.example.junit\_demo;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class PaymentServiceTest {

@Test

void testInteractionOrder() {

PaymentApi mockApi = mock(PaymentApi.class);

PaymentService service = new PaymentService(mockApi);

service.processPayment();

InOrder inOrder = inOrder(mockApi);

inOrder.verify(mockApi).initiatePayment();

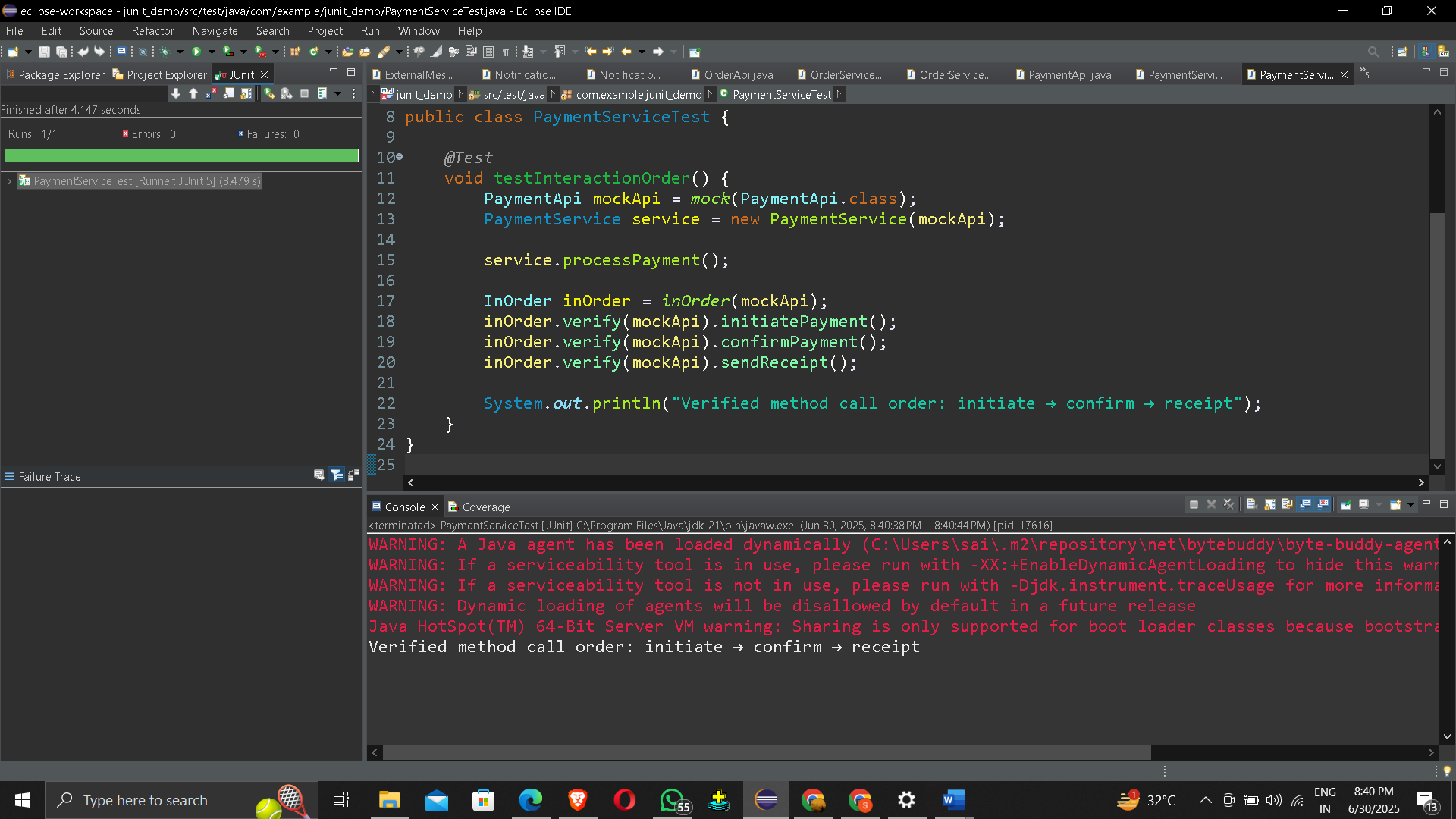
inOrder.verify(mockApi).confirmPayment();

inOrder.verify(mockApi).sendReceipt();

System.out.println("Verified method call order: initiate → confirm → receipt");

}

}



# Exercise 7: Handling Void Methods with Exceptions

Scenario:

You need to test a void method that throws an exception.

Steps:

1. Create a mock object.
2. Stub the void method to throw an exception.
3. Verify the interaction.

package com.example.junit\_demo;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class LogServiceTest {

@Test

void testVoidMethodThrowsException() {

Logger mockLogger = mock(Logger.class);

// Stub the void method to throw an exception

doThrow(new RuntimeException("Logging failed"))

.when(mockLogger).log("CRITICAL");

LogService service = new LogService(mockLogger);

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

service.writeLog("CRITICAL");

});

assertEquals("Logging failed", exception.getMessage());

verify(mockLogger).log("CRITICAL");

}

}

