**MOCKITO EXERCISES**

**Exercise 1: Mocking and Stubbing**

**// ExternalApi.java**

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

**// MyService.java**

package org.example;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**// MyServiceTest.java**

package org.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);

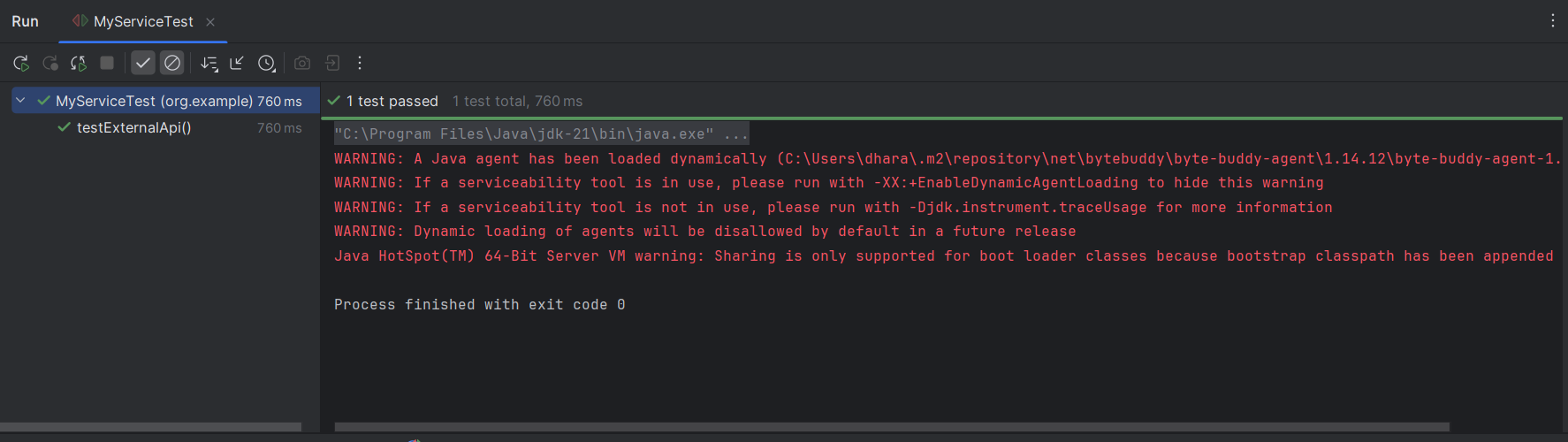
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**Output:**



**Exercise 2: Verifying Interactions**

**// ExternalApi.java**

package org.example;

public interface ExternalApi {

String getData();

}

**// MyService.java**

package org.example;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData(); // This method call will be verified in test

}

}

**// MyServiceTest.java**

package org.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);

// Call the method

service.fetchData();

verify(mockApi).getData();

}

}

**// pom.xml**

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

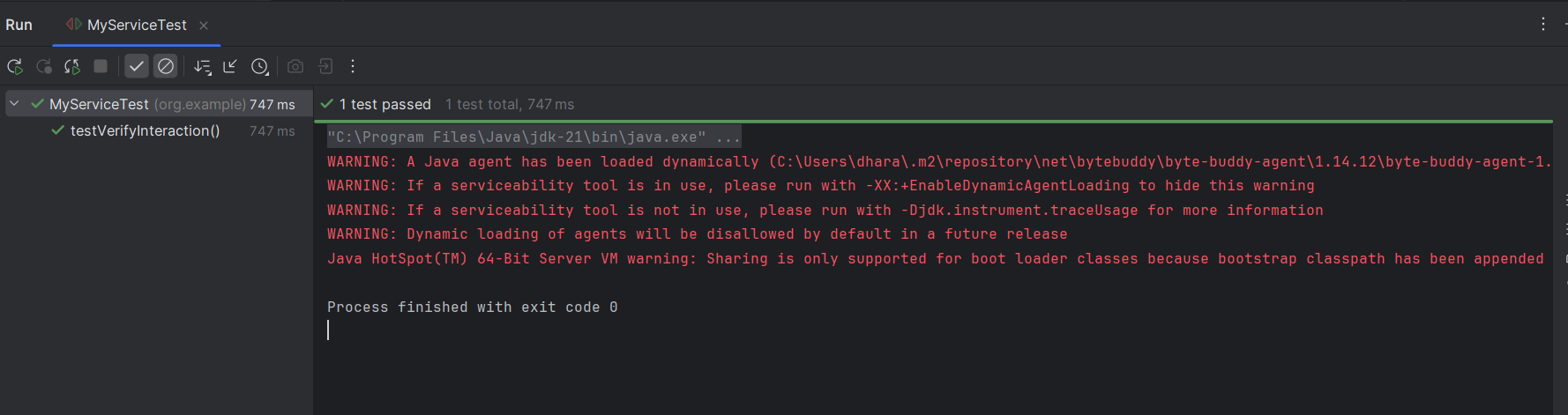
<version>5.11.0</version>

<scope>test</scope>

</dependency>

</dependencies>

**Output:**



**Exercise 3: Argument Matching**

**// UserNotifier.java**

package org.example;

public interface UserNotifier {

void sendNotification(String userId, String message);

}

**// NotificationService.java**

package org.example;

public class NotificationService {

private final UserNotifier notifier;

public NotificationService(UserNotifier notifier) {

this.notifier = notifier;

}

public void notifyUser(String userId) {

notifier.sendNotification(userId, "Welcome, " + userId + "!");

}

}

**// NotificationServiceTest.java**

package org.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

public class NotificationServiceTest {

@Test

public void testArgumentMatching() {

UserNotifier mockNotifier = mock(UserNotifier.class);

NotificationService service = new NotificationService(mockNotifier);

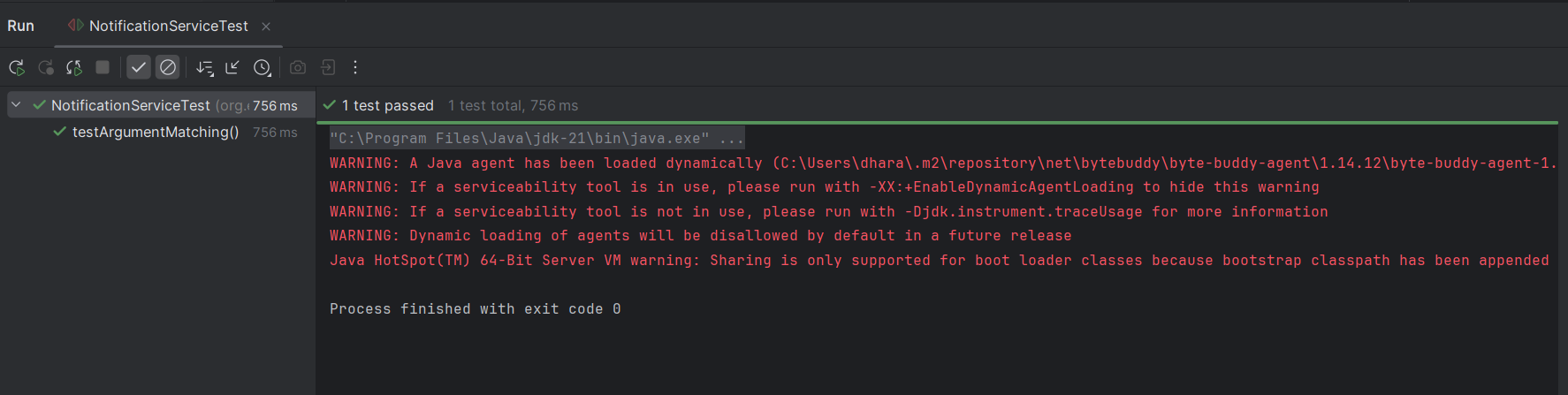
service.notifyUser("user123");

verify(mockNotifier).sendNotification(eq("user123"), contains("Welcome"));

}

}

**Output:**



**Exercise 4: Handling Void Methods**

**// UserNotifier.java**

package org.example;

public interface UserNotifier {

void sendNotification(String userId);

}

**// NotificationService.java**

package org.example;

public class NotificationService {

private final UserNotifier notifier;

public NotificationService(UserNotifier notifier) {

this.notifier = notifier;

}

public void triggerNotification(String userId) {

notifier.sendNotification(userId);

}

}

**// NotificationServiceTest.java**

package org.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class NotificationServiceTest {

@Test

public void testVoidMethodInteraction() {

UserNotifier mockNotifier = mock(UserNotifier.class);

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

NotificationService service = new NotificationService(mockNotifier);

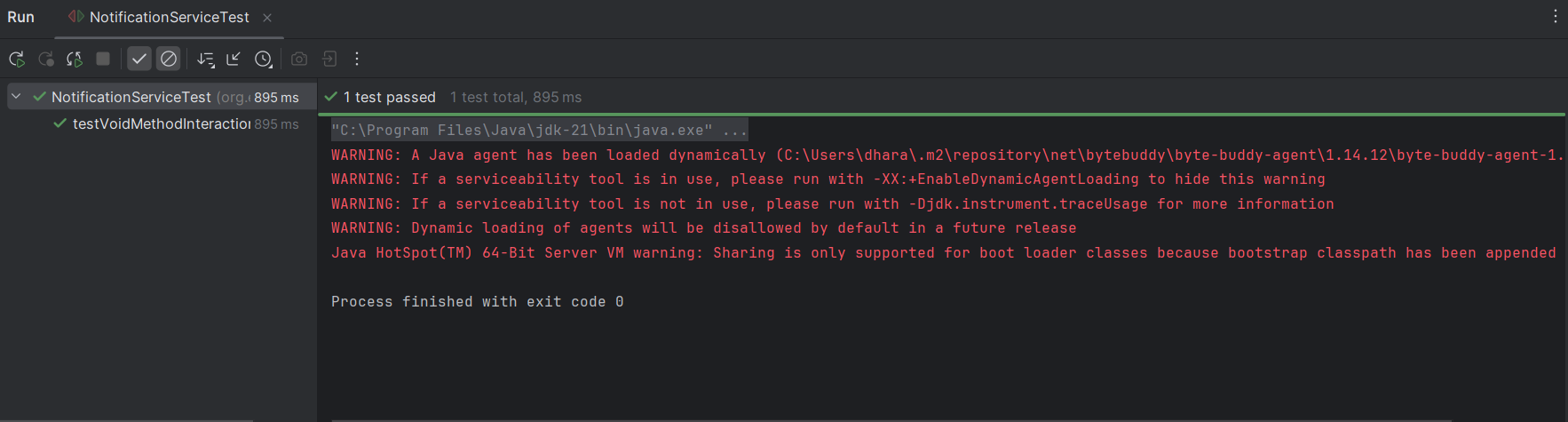
service.triggerNotification("user123");

verify(mockNotifier).sendNotification("user123");

}

}

**Output:**



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

**// ExternalApi.java**

package org.example;

public interface ExternalApi {

String getStatus();

}

**// PollingService.java**

package org.example;

public class PollingService {

private final ExternalApi api;

public PollingService(ExternalApi api) {

this.api = api;

}

public String waitForCompletion() {

String status;

do {

status = api.getStatus();

} while (!"DONE".equals(status));

return status;

}

}

**// PollingServiceTest.java**

package org.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class PollingServiceTest {

@Test

public void testMultipleReturnsFromMock() {

ExternalApi mockApi = mock(ExternalApi.class);

when(mockApi.getStatus())

.thenReturn("PENDING")

.thenReturn("PENDING")

.thenReturn("DONE");

PollingService service = new PollingService(mockApi);

String result = service.waitForCompletion();

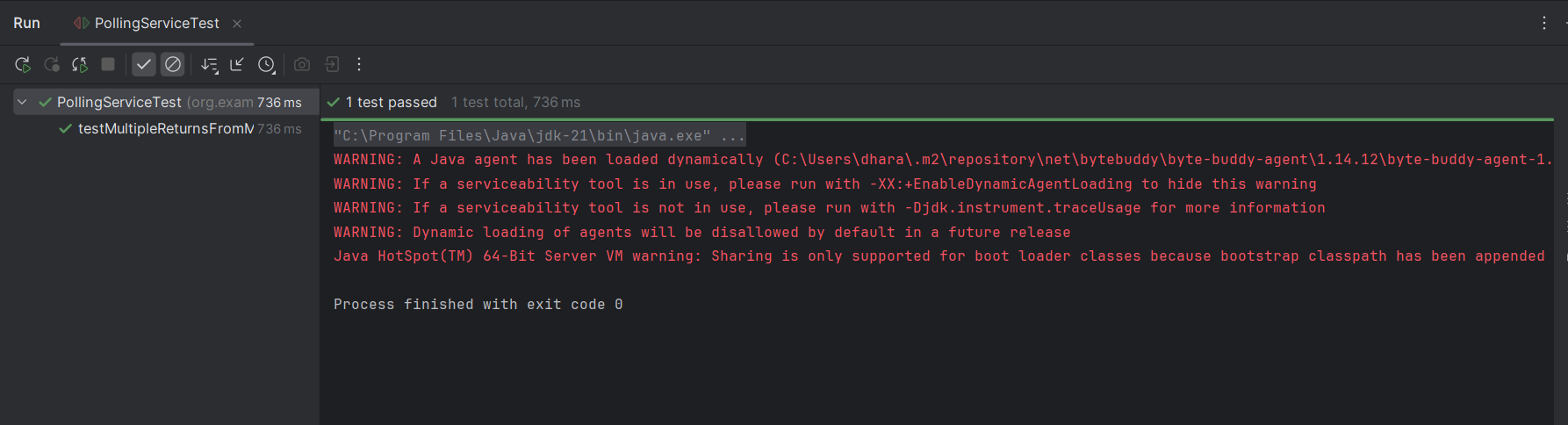
assertEquals("DONE", result);

verify(mockApi, times(3)).getStatus();

}

}

**Output:**



**Exercise 6: Verifying Interaction Order**

**// ReportGenerator.java**

package org.example;

public interface ReportGenerator {

void initialize();

void generateReport();

}

**// ReportService.java**

package org.example;

public class ReportService {

private final ReportGenerator generator;

public ReportService(ReportGenerator generator) {

this.generator = generator;

}

public void process() {

generator.initialize();

generator.generateReport();

}

}

**// ReportServiceTest.java**

package org.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class ReportServiceTest {

@Test

public void testMethodCallOrder() {

ReportGenerator mockGenerator = mock(ReportGenerator.class);

ReportService service = new ReportService(mockGenerator);

service.process();

InOrder inOrder = inOrder(mockGenerator);

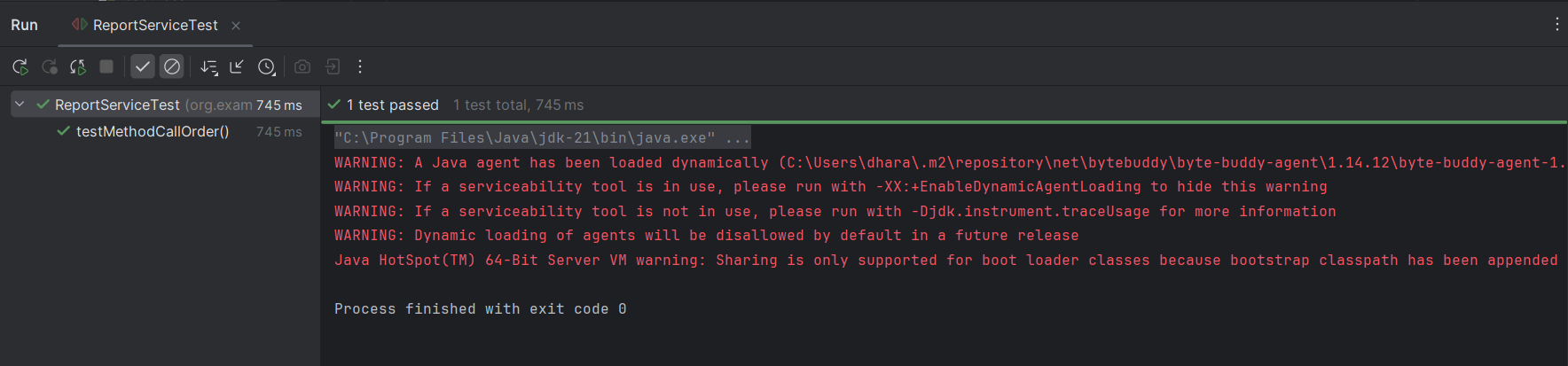
inOrder.verify(mockGenerator).initialize();

inOrder.verify(mockGenerator).generateReport();

}

}

**Output:**



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

**// UserNotifier.java**

package org.example;

public interface UserNotifier {

void sendEmail(String userId);

}

**// EmailService.java**

package org.example;

public class EmailService {

private final UserNotifier notifier;

public EmailService(UserNotifier notifier) {

this.notifier = notifier;

}

public void notifyUser(String userId) {

notifier.sendEmail(userId);

}

}

**// EmailServiceTest.java**

package org.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

import org.mockito.Mockito;

public class EmailServiceTest {

@Test

public void testVoidMethodThrowsException() {

UserNotifier mockNotifier = mock(UserNotifier.class);

doThrow(new RuntimeException("Email failed")).when(mockNotifier).sendEmail("user123");

EmailService service = new EmailService(mockNotifier);

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

service.notifyUser("user123");

});

assertEquals("Email failed", thrown.getMessage());

verify(mockNotifier).sendEmail("user123");

}

}

**Output:**

