**MOCKITO ADVANCED EXERCISES**

**Exercise 1: Mocking Databases and Repositories**

**// Repository.java**

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

**// Service.java**

package org.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;  
 }  
}

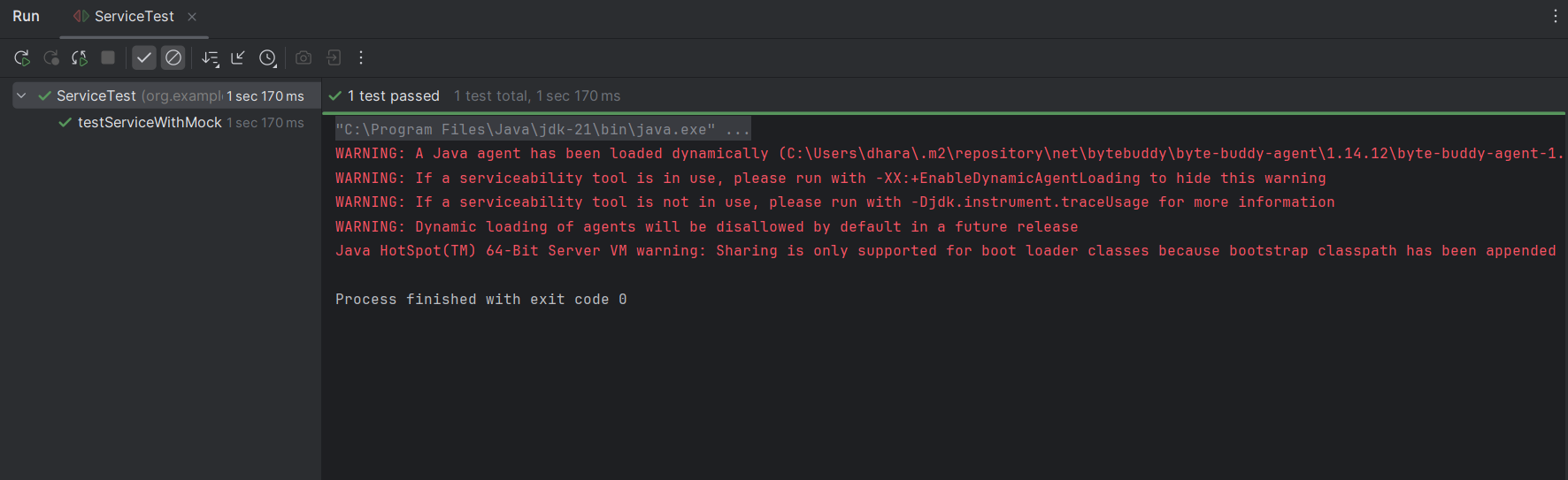
**// ServiceTest.java**

package org.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() {  
 // Create mock  
 Repository mockRepository = *mock*(Repository.class);  
  
 // Stub method  
 *when*(mockRepository.getData()).thenReturn("Mock Data");  
  
 // Use in service  
 Service service = new Service(mockRepository);  
 String result = service.processData();  
  
 // Assert and verify  
 *assertEquals*("Processed Mock Data", result);  
 *verify*(mockRepository).getData(); // optional, to ensure it was called  
 }  
}

**// pom.xml**

<?xml version="1.0" encoding="UTF-8"?>  
<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>org.example</groupId>  
 <artifactId>Mockito\_Advanced</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>21</maven.compiler.source>  
 <maven.compiler.target>21</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>5.11.0</version> <!-- latest as of 2024 -->  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>5.10.2</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.junit.platform</groupId>  
 <artifactId>junit-platform-suite</artifactId>  
 <version>1.10.0</version>  
 <scope>test</scope>  
 </dependency>  
  
 </dependencies>  
  
</project>

**Output:**



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

**// RestClient.java**

package org.example;

public interface RestClient {

String getResponse(); // Simulates a GET API call

}

**// ApiService.java**

package org.example;

public class ApiService {

private final RestClient restClient;

public ApiService(RestClient restClient) {

this.restClient = restClient;

}

public String fetchData() {

String response = restClient.getResponse();

return "Fetched " + response;

}

}

**// ApiServiceTest.java**

package org.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() {

// Create mock REST client

RestClient mockRestClient = mock(RestClient.class);

// Stub the API call

when(mockRestClient.getResponse()).thenReturn("Mock Response");

// Inject into service

ApiService apiService = new ApiService(mockRestClient);

// Call method and verify result

String result = apiService.fetchData();

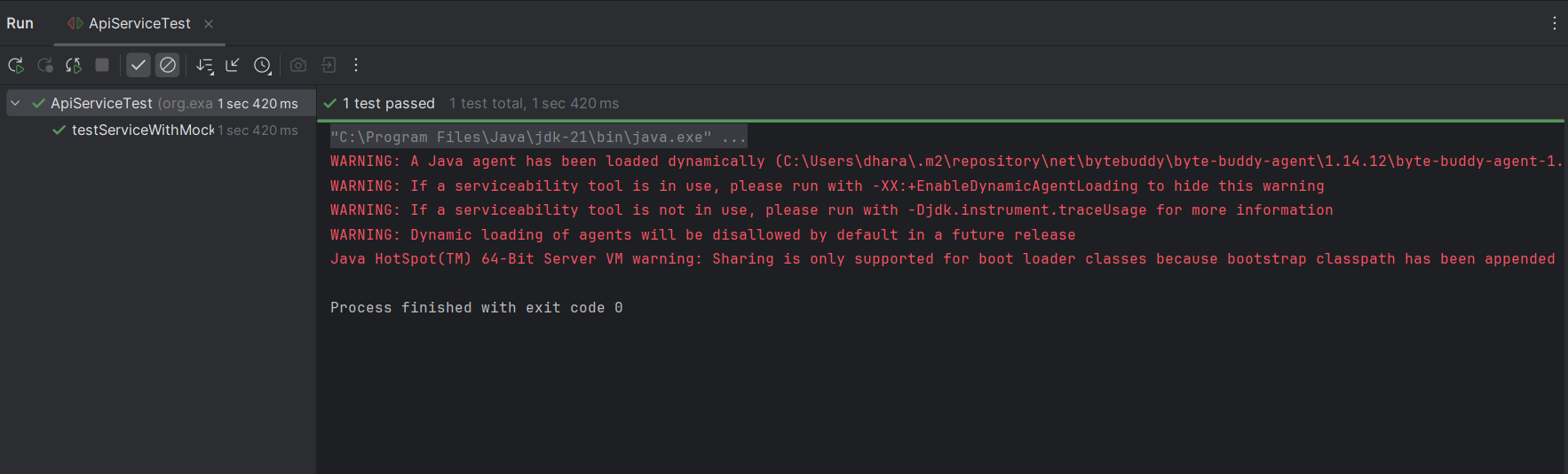
assertEquals("Fetched Mock Response", result);

verify(mockRestClient).getResponse(); // Optional: Verify interaction

}

}

**Output:**



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

**// FileReader.java**

package org.example;

public interface FileReader {

String read();

}

**// FileWriter.java**

package org.example;

public interface FileWriter {

void write(String content);

}

**// FileService.java**

package org.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 processedContent = "Processed " + content;

fileWriter.write(processedContent);

return processedContent;

}

}

**// FileServiceTest.java**

package org.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() {

// Mock the dependencies

FileReader mockFileReader = mock(FileReader.class);

FileWriter mockFileWriter = mock(FileWriter.class);

// Stub the read method

when(mockFileReader.read()).thenReturn("Mock File Content");

// Inject into the service

FileService fileService = new FileService(mockFileReader, mockFileWriter);

// Call the method

String result = fileService.processFile();

// Assert and verify

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

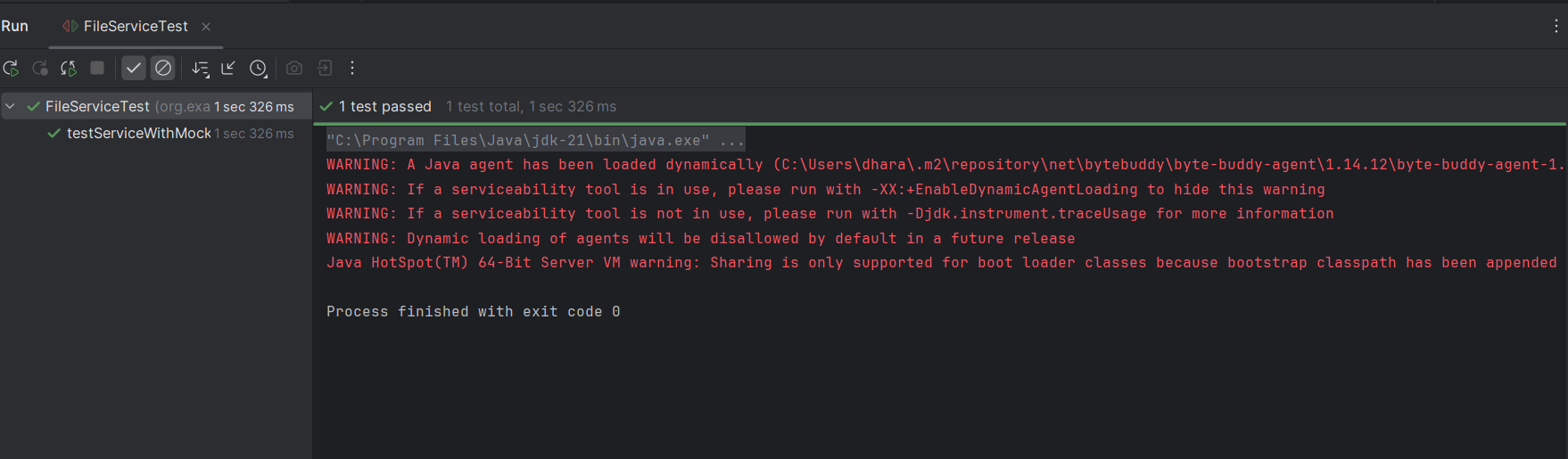
verify(mockFileReader).read();

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

}

}

**Output:**



**Exercise 4: Mocking Network Interactions**

**// NetworkClient.java**

package org.example;

public interface NetworkClient {

String connect();

}

**// NetworkService.java**

package org.example;

public class NetworkService {

private final NetworkClient networkClient;

public NetworkService(NetworkClient networkClient) {

this.networkClient = networkClient;

}

public String connectToServer() {

String connection = networkClient.connect();

return "Connected to " + connection;

}

}

**// NetworkServiceTest.java**

package org.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() {

// Create a mock network client

NetworkClient mockNetworkClient = mock(NetworkClient.class);

// Stub the connect method

when(mockNetworkClient.connect()).thenReturn("Mock Connection");

// Use mock in service

NetworkService networkService = new NetworkService(mockNetworkClient);

String result = networkService.connectToServer();

// Assert and verify

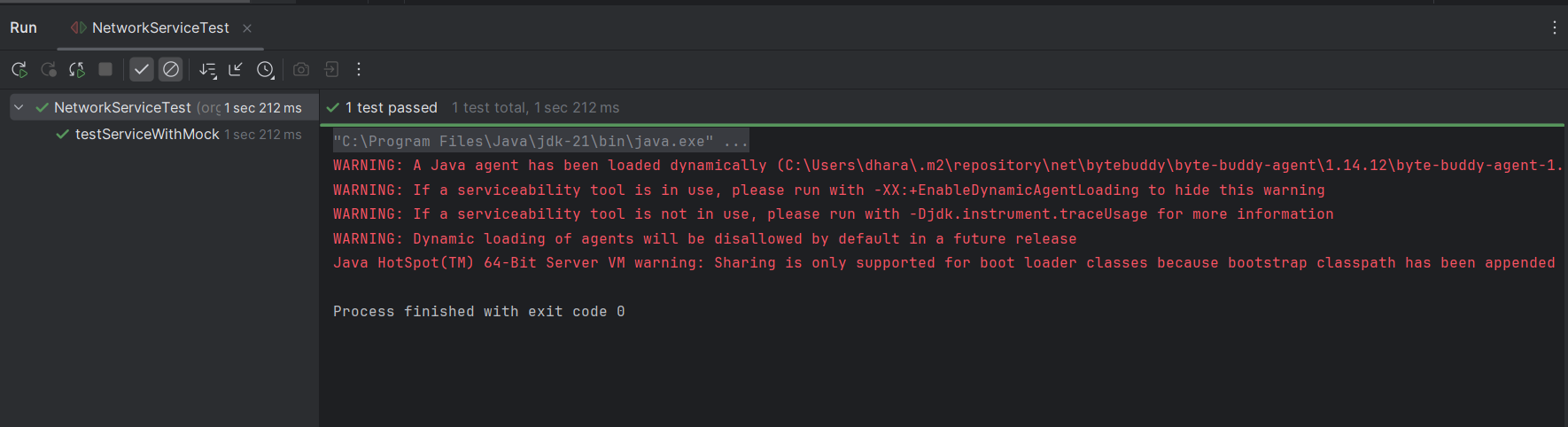
assertEquals("Connected to Mock Connection", result);

verify(mockNetworkClient).connect(); // optional: ensure connect() was called

}

}

**Output:**



**Exercise 5: Mocking Multiple Return Values**

**// Repository.java**

package org.example;

public interface Repository {

String getData();

}

**// Service.java**

package org.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 org.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() {

// Mock repository

Repository mockRepository = mock(Repository.class);

// Stub multiple return values

when(mockRepository.getData())

.thenReturn("First Mock Data")

.thenReturn("Second Mock Data");

// Use in service

Service service = new Service(mockRepository);

// Call twice and capture results

String firstResult = service.processData();

String secondResult = service.processData();

// Assertions

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

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

// Verify two invocations

verify(mockRepository, times(2)).getData();

}

}

**Output:**

