**WEEK-2**

**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

**Code:**

**ExternalApi.java**

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

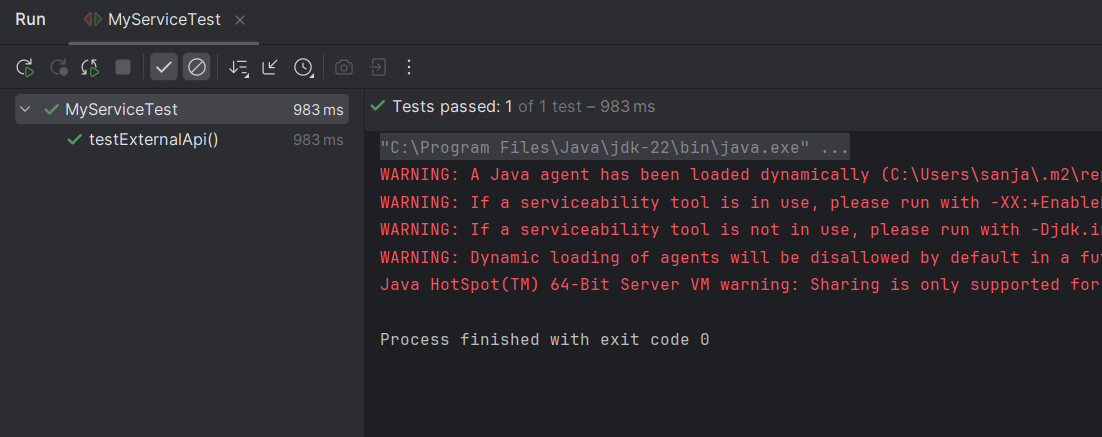
**MyService.java**

package com.example;  
  
public class MyService {  
 private final ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData();  
 }  
}

**MyServiceTest.java**

import com.example.ExternalApi;  
import com.example.MyService;  
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() {  
 // 1. Create mock  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 // 2. Stub method  
 *when*(mockApi.getData()).thenReturn("Mock Data");  
  
 // 3. Use mock in service  
 MyService service = new MyService(mockApi);  
 String result = service.fetchData();  
  
 // 4. Assert  
 *assertEquals*("Mock Data", result);  
 }  
}

**Output:**

****

**Exercise 2: Verifying Interactions**

**Code:**

**ExternalApi.java**

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

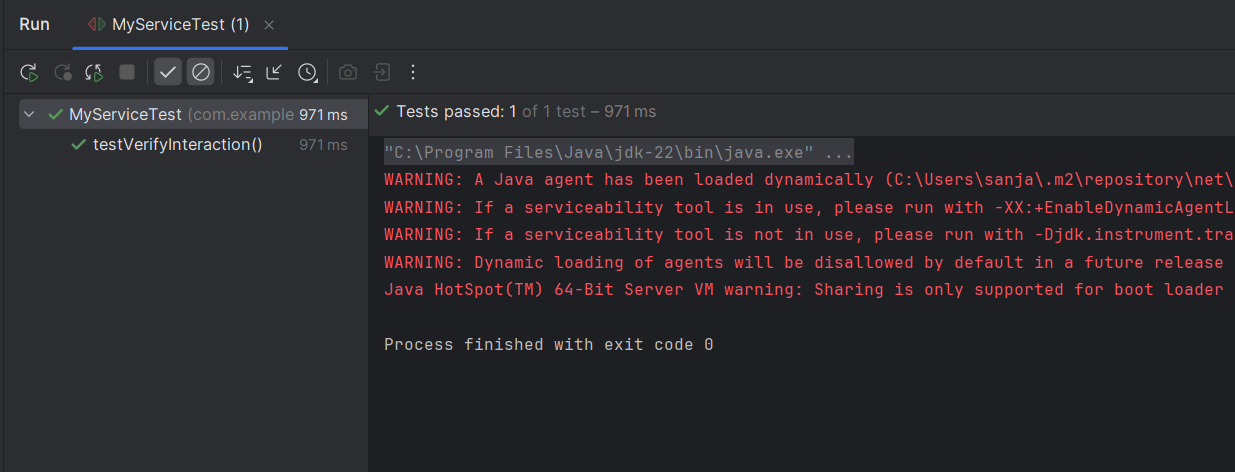
**MyService.java**

package com.example;  
  
public class MyService {  
 private final ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData();  
 }  
}

**MyServiceTest.java**

package com.example;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testVerifyInteraction() {  
 // Step 1: Create mock  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 // Step 2: Inject into service and call method  
 MyService service = new MyService(mockApi);  
 service.fetchData();  
  
 // Step 3: Verify that mockApi.getData() was called  
 *verify*(mockApi).getData();  
 }  
}

**Output:**

****

**Exercise 3: Argument Matching**

**Code:**

**ExternalApi.java**

package com.example;  
  
public interface ExternalApi {  
 void sendData(String message);  
}

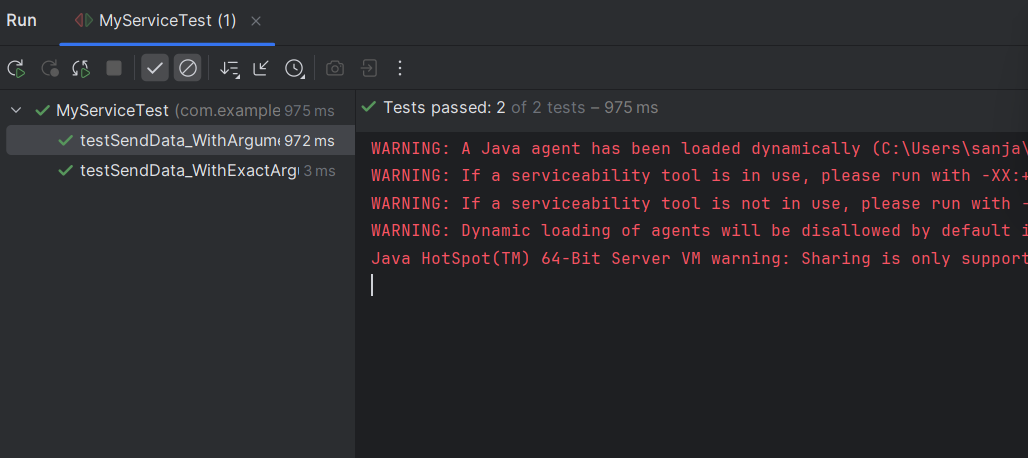
**MyService.java**

package com.example;  
  
public class MyService {  
 private final ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public void processAndSend(String input) {  
 api.sendData("Processed: " + input);  
 }  
}

**MyServiceTest.java**

package com.example;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.mockito.ArgumentMatchers.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testSendData\_WithExactArgument() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 MyService service = new MyService(mockApi);  
  
 service.processAndSend("Sanjai");  
  
 *verify*(mockApi).sendData("Processed: Sanjai");  
 }  
  
 @Test  
 public void testSendData\_WithArgumentMatcher() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 MyService service = new MyService(mockApi);  
  
 service.processAndSend("Cognizant");  
  
 *verify*(mockApi).sendData(*startsWith*("Processed:"));  
 }  
}

**Output:**

****

**Exercise 4: Handling Void Methods**

**Code:**

**Logger.java**

package com.example;  
  
public interface Logger {  
 void log(String message);  
}

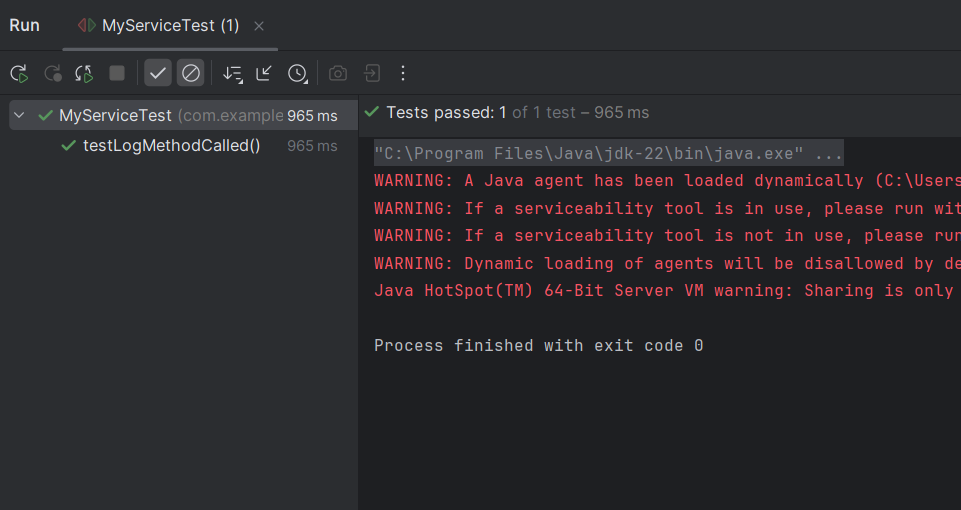
**MyService.java**

package com.example;  
  
public class MyService {  
 private final Logger logger;  
  
 public MyService(Logger logger) {  
 this.logger = logger;  
 }  
  
 public void doSomething() {  
 logger.log("Action completed");  
 }  
}

**MyServiceTest.java**

package com.example;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testLogMethodCalled() {  
 Logger mockLogger = *mock*(Logger.class);  
  
 *doNothing*().when(mockLogger).log(*anyString*());  
  
 MyService service = new MyService(mockLogger);  
 service.doSomething();  
  
 *verify*(mockLogger).log("Action completed");  
 }  
}

**Output:**

****

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

**Code:**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getStatus();

}

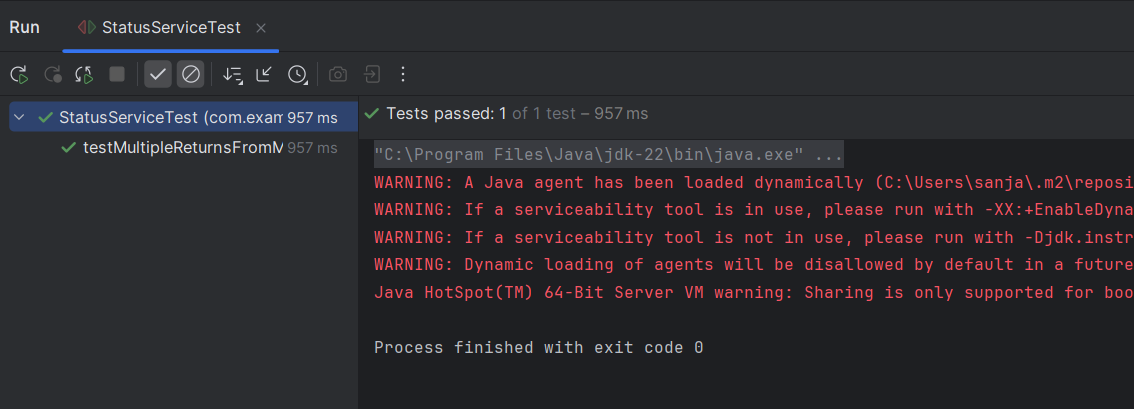
**StatusService.java**

package com.example;  
  
public class StatusService {  
 private final ExternalApi api;  
  
 public StatusService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String checkStatus() {  
 return api.getStatus();  
 }  
}

**StatusServiceTest.java**

package com.example;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class StatusServiceTest {  
  
 @Test  
 public void testMultipleReturnsFromMock() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 *when*(mockApi.getStatus())  
 .thenReturn("Pending")  
 .thenReturn("Processing")  
 .thenReturn("Completed");  
  
 StatusService service = new StatusService(mockApi);  
  
 *assertEquals*("Pending", service.checkStatus());  
 *assertEquals*("Processing", service.checkStatus());  
 *assertEquals*("Completed", service.checkStatus());  
 }  
}

**Output:**

****

**Exercise 6: Verifying Interaction Order**

**Code:**

**ExternalApi.java**

package com.example;  
  
public interface ExternalApi {  
 void connect();  
 void fetchData();  
 void disconnect();  
}

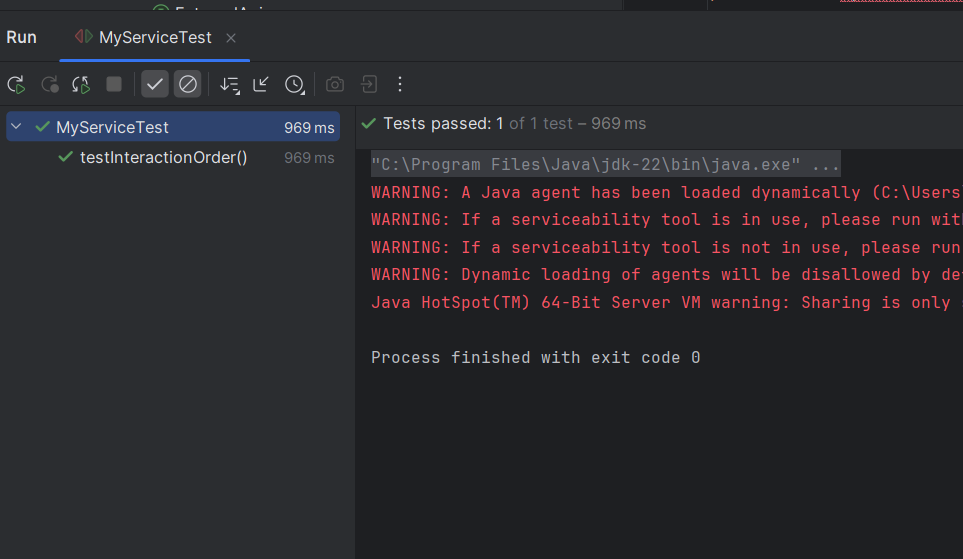
**MyService.java**

package com.example;  
  
public class MyService {  
 private final ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public void run() {  
 api.connect();  
 api.fetchData();  
 api.disconnect();  
 }  
}

**MyServiceTest.java**

import com.example.MyService;  
import com.example.ExternalApi;  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import org.mockito.InOrder;  
  
public class MyServiceTest {  
  
 @Test  
 public void testInteractionOrder() {   
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 MyService service = new MyService(mockApi);  
 service.run();  
 InOrder inOrder = *inOrder*(mockApi);  
  
 inOrder.verify(mockApi).connect();  
 inOrder.verify(mockApi).fetchData();  
 inOrder.verify(mockApi).disconnect();  
 }  
}

**Output:**

****

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

**Code:**

**Notifier.java**

package com.example;  
  
public interface Notifier {  
 void sendNotification(String message);  
}

**AlertService.java**

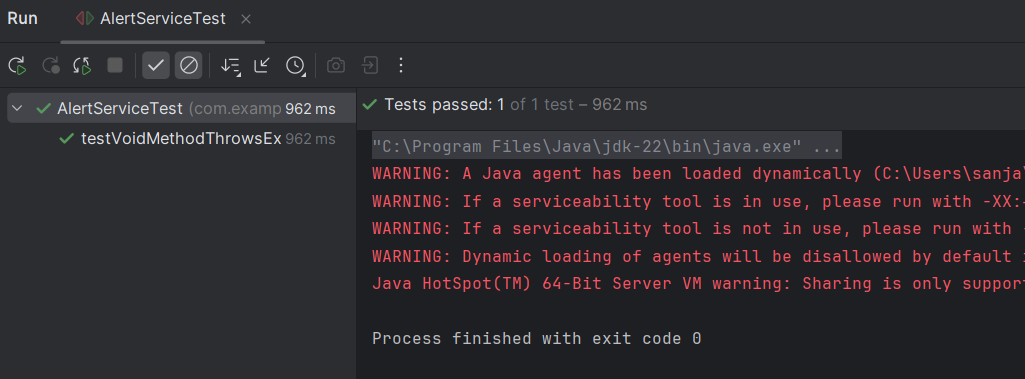
package com.example;  
  
public class AlertService {  
 private final Notifier notifier;  
  
 public AlertService(Notifier notifier) {  
 this.notifier = notifier;  
 }  
  
 public void alert(String message) {  
 notifier.sendNotification(message);  
 }  
}

**AlertServiceTest.java**

package com.example;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class AlertServiceTest {  
  
 @Test  
 public void testVoidMethodThrowsException() {  
 Notifier mockNotifier = *mock*(Notifier.class);  
  
 *doThrow*(new RuntimeException("Notification failed"))  
 .when(mockNotifier).sendNotification("CRITICAL");

AlertService service = new AlertService(mockNotifier);  
  
 RuntimeException exception = *assertThrows*(RuntimeException.class, () -> {  
 service.alert("CRITICAL");  
 });  
  
 *assertEquals*("Notification failed", exception.getMessage());  
  
 *verify*(mockNotifier).sendNotification("CRITICAL");  
 }  
}

**Output:**

****