

## JUnit Testing Exercises

### 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:

```
import static org.mockito.Mockito.*;
```

```
import org.junit.jupiter.api.Test;
```

```
import org.mockito.Mockito;
```

```
public class MyServiceTest {
```

```
    @Test
```

```
    public void testVerifyInteraction() {
```

```
        ExternalApi mockApi = Mockito.mock(ExternalApi.class);
```

```
        MyService service = new MyService(mockApi);
```

```
        service.fetchData();
```

```
        verify(mockApi).getData();
```

```
    }
```

```
}
```

#### **pom.xml**

```
<dependency>
```

```
    <groupId>org.mockito</groupId>
```

```
    <artifactId>mockito-core</artifactId>
```

```
    <version>4.11.0</version>
```

```
    <scope>test</scope>
```

```
</dependency>
```

### **ExternalApi.java**

```
package com.example;
```

```
public interface ExternalApi {  
    String getData();  
}
```

### **MyService.java**

```
package com.example;
```

```
public class MyService {  
    private ExternalApi api;  
  
    public MyService(ExternalApi api) {  
        this.api = api;  
    }  
  
    public String fetchData() {  
        return api.getData();  
    }  
}
```

### **MyServiceTest.java**

```
package com.example;
```

```
import org.junit.Test;  
import static org.mockito.Mockito.*;
```

```
public class MyServiceTest {  
  
    @Test  
    public void testVerifyInteraction() {  
        // Step 1: Create a mock  
        ExternalApi mockApi = mock(ExternalApi.class);
```

```

// Step 2: Pass mock to service and call method
MyService service = new MyService(mockApi);
service.fetchData();

// Step 3: Verify interaction
verify(mockApi).getData();
}
}

```

## Explanation of Key Concepts Used

In this test, we use `Mockito.verify()` to check whether a specific method was called during the test. Instead of only checking results, this focuses on verifying behavior and interactions with dependencies. In real-world cases, this is useful for ensuring the service logic actually performs its intended actions (like saving, fetching, or logging).

## OUTPUT:

The screenshot shows an IDE with a JUnit test runner on the left and a Java code editor on the right. The test runner indicates that the test 'com.example.MyServiceTest' passed successfully after 0.488 seconds, with 1/1 runs, 0 errors, and 0 failures. The code editor shows the following Java code:

```

1 package com.example;
2
3 import org.junit.Test;
4 import static org.mockito.Mockito.*;
5 import static org.junit.Assert.*;
6
7 public class MyServiceTest {
8
9     @Test
10    public void testExternalApi() {
11        // Step 1: Create mock
12        ExternalApi mockApi = mock(ExternalApi.class);
13
14        // Step 2: Stub method
15        when(mockApi.getData()).thenReturn("Mock Data");
16    }
17 }

```