



Xideral

Java Academy

Week 4-Day 4

Mockito

Presented by:

Edgar Itzak Sánchez Rogers

Monterrey Nuevo León México at 6 September 2024

## Introduction:

Mockito is a java based mocking framework, used in conjunction with other testing frameworks such as JUnit and TestNG. It internally uses Java Reflection API and allows to create objects of a service. A mock object returns a dummy data and avoids external dependencies. It simplifies the development of tests by mocking external dependencies and apply the mocks into the code under test.



## Benefits of Mockito

- **No Handwriting** – No need to write mock objects on your own.
- **Refactoring Safe** – Renaming interface method names or reordering parameters will not break the test code as Mocks are created at runtime.
- **Return value support** – Supports return values.
- **Exception support** – Supports exceptions.
- **Order check support** – Supports check on order of method calls.
- **Annotation support** – Supports creating mocks using annotation.

## Test example:

### Pom.xml

```
9      <!-- https://mvnrepository.com/artifact/org.mockito/mockito-core -->
0      <dependency>
1          <groupId>org.mockito</groupId>
2          <artifactId>mockito-core</artifactId>
3          <version>5.12.0</version>
4          <scope>test</scope>
5      </dependency>
6
7      <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api -->
8      <dependency>
9          <groupId>org.junit.jupiter</groupId>
0          <artifactId>junit-jupiter-api</artifactId>
1          <version>5.10.2</version>
2          <scope>test</scope>
3      </dependency>
4
5      <dependency>
6          <groupId>org.junit.jupiter</groupId>
7          <artifactId>junit-jupiter-engine</artifactId>
8          <version>5.10.2</version>
9          <scope>test</scope>
```

### Interface:

```
1 package com.edgaritzak.mockito;
2
3 public interface ExternPasswordVerifier {
4
5     boolean verifyPassword(String password, boolean lowercase, boolean uppercase, boolean digit);
6 }
```













### Verifier:

```
1 package com.edgaritzak.mockito;
2
3 public class Verifier {
4
5     private ExternPasswordVerifier externService;
6
7     public Verifier(ExternPasswordVerifier externService) {
8         this.externService = externService;
9     }
10
11     public boolean verifyPassword(String password, boolean lowercase, boolean uppercase, boolean digit) {
12         return externService.verifyPassword(password, lowercase, uppercase, digit);
13     }
14 }
```

## TestDemo:

```
1 package com.edgaritzak.mockito;
2
3 import static org.junit.jupiter.api.Assertions.*;
13
14 class TestDemo {
15
16     @Mock
17     ExternPasswordVerifier externService;
18     Verifier verifier;
19
20     @BeforeEach
21     void setUp() throws Exception {
22         externService = mock(ExternPasswordVerifier.class);
23         verifier = new Verifier(externService);
24     }
25
26     @Test
27     void testPasswordVerifierNoUppercase() {
28         when(externService.verifyPassword("a1", true, true, true)).thenReturn(false);
29
30         boolean result = verifier.verifyPassword("a1", true, true, true);
31         assertEquals(false, result);
32         verify(externService).verifyPassword("a1", true, true, true);
33     }
34
35     @Test
36     void testPasswordVerifierNoLowercase() {
37         when(externService.verifyPassword("1A", true, true, true)).thenReturn(false);
38         boolean result = verifier.verifyPassword("1A", true, true, true);
39         assertEquals(false, result);
40         verify(externService).verifyPassword("1A", true, true, true);
41     }
42
43     @Test
44     void testPasswordVerifierNoDigit() {
45         when(externService.verifyPassword("Aa", true, true, true)).thenReturn(false);
46         boolean result = verifier.verifyPassword("Aa", true, true, true);
47         assertEquals(false, result);
48         verify(externService).verifyPassword("Aa", true, true, true);
49     }
50
51     @Test
52     void testPasswordVerifierValidPassword() {
53         when(externService.verifyPassword("1Aa", true, true, true)).thenReturn(true);
54         boolean result = verifier.verifyPassword("1Aa", true, true, true);
55         assertEquals(true, result);
56         verify(externService).verifyPassword("1Aa", true, true, true);
57     }
58 }
```

## Coverage:

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
mockito-example	 100.0 %	178	0	178
src/main/java	 100.0 %	14	0	14
com.edgaritzak.mockito	 100.0 %	14	0	14
src/test/java	 100.0 %	164	0	164
com.edgaritzak.mockito	 100.0 %	164	0	164
TestDemo.java	 100.0 %	164	0	164
TestDemo	 100.0 %	164	0	164
setUp()	 100.0 %	13	0	13
testPasswordVerifierNoDigit()	 100.0 %	37	0	37
testPasswordVerifierNoLowercase()	 100.0 %	37	0	37
testPasswordVerifierNoUppercase()	 100.0 %	37	0	37
testPasswordVerifierValidPassword()	 100.0 %	37	0	37

## Conclusion:

Mockito offers several advantages for Java testing:

Mockito provides an intuitive and easy-to-use API, which simplifies the process of creating and managing mocks. Mockito allows developers to write tests faster and with less effort, allows mocks of interfaces and abstract classes to be created, making it easier to test components in isolation without relying on real or external implementations.

Mockito Provides advanced functionality for configuring mock behavior and verifying interactions, helping to ensure that methods are called with the correct parameters and in the expected order. Mockito Helps ensure that unit tests focus only on the behavior of the unit of code in question, without relying on the functionality of other parts of the system.

## References:

[1] [Mockito Tutorial | DigitalOcean](#)

[2] [Mockito - Overview \(tutorialspoint.com\)](#)