Let's consider a simple scenario where we have a `Calculator` class that depends on a `MathService` for performing calculations. We'll use Mockito to mock the `MathService` and test the `Calculator` class.

Step 1: Add Mockito Dependency

Add the following dependency to your `pom.xml` if you're using Maven:

```xml

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>3.11.2</version>

<scope>test</scope>

</dependency>

```

Step 2: Create Classes

Create the classes you'll be testing:

1. `MathService` (Dependency class):

```java

public class MathService {

public int add(int a, int b) {

return a + b;

}

}

```

2. `Calculator` (Class under test):

```java

public class Calculator {

private MathService mathService;

public Calculator(MathService mathService) {

this.mathService = mathService;

}

public int addNumbers(int a, int b) {

return mathService.add(a, b);

}

}

```

Step 3: Write the Test Class

Create a test class for the `Calculator` class, named `CalculatorTest`.

```java

import org.junit.Test;

import org.mockito.Mockito;

import static org.junit.Assert.assertEquals;

public class CalculatorTest {

@Test

public void testAddNumbers() {

// Create a mock of MathService

MathService mathServiceMock = Mockito.mock(MathService.class);

// Define mock behavior for the add method

Mockito.when(mathServiceMock.add(2, 3)).thenReturn(5);

// Create an instance of Calculator with the mock MathService

Calculator calculator = new Calculator(mathServiceMock);

// Call the method under test

int result = calculator.addNumbers(2, 3);

// Verify the result

assertEquals(5, result);

// Verify that the add method of MathService was called with the correct arguments

Mockito.verify(mathServiceMock, Mockito.times(1)).add(2, 3);

}

}

```

In this test class, we're using Mockito to:

- Create a mock of the `MathService` class.

- Define the behavior of the `add` method of the mock.

- Create an instance of the `Calculator` class, injecting the mock `MathService`.

- Call the `addNumbers` method of the `Calculator` and verify the result.

- Verify that the `add` method of the mock `MathService` was called with the correct arguments.

Step 4: Run the Tests

Run your test using your preferred test runner (e.g., JUnit). The test should pass, demonstrating how to use Mockito to test the behavior of your classes while isolating dependencies.

This example demonstrates a basic use case of Mockito. You can use similar principles for more complex scenarios involving different methods, return types, and verification checks. Remember to adjust the package names, class names, and test logic to match your project's structure.