# Mockito exercises

## Source code: Here

# Exercise 1: Mocking and Stubbing

Testing with **Mockito** means mocking the **service classes** (here our external API) and **stubbing their methods** (defining how they should behave when called) to isolate and test the logic of the class under test.

# Code:

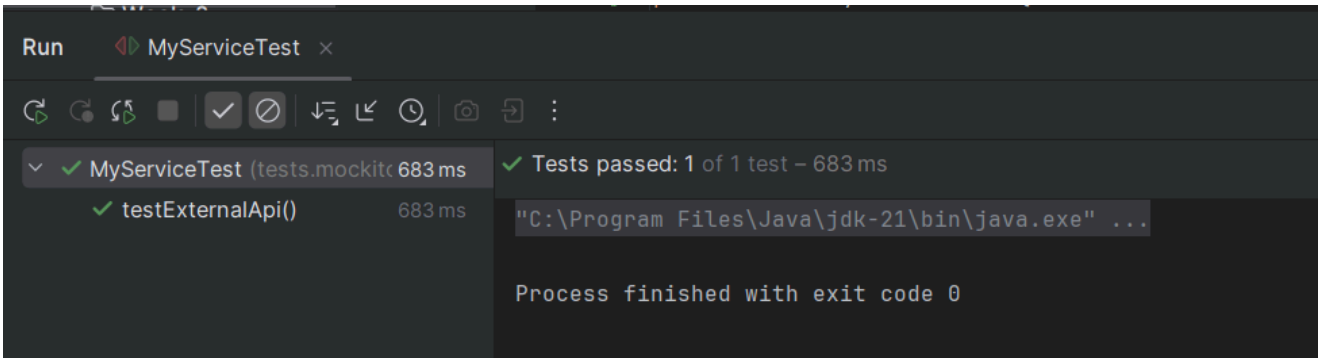
```
package exercises.mokito;

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

```
package exercises.mokito;

public class MyService {
    private ExternalApi api;

    public MyService(ExternalApi api) {
        this.api = api;
    }

    public String fetchData() {
        return api.getData();
    }
}
```

```
package tests.mockito;

import exercises.mokito.ExternalApi;
import exercises.mokito.MyService;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;

import static org.mockito.Mockito.*;

public class MyServiceTest {
```

```
    @Test
    public void testExternalApi() {
        ExternalApi mockApi = Mockito.mock(ExternalApi.class);
        when(mockApi.getData()).thenReturn("Mock Data");
        MyService service = new MyService(mockApi);
        String result = service.fetchData();
        Assertions.assertEquals("Mock Data", result);
    }
}
```

## Results:




---

# Exercise 2: Verifying Interactions

## Type-1

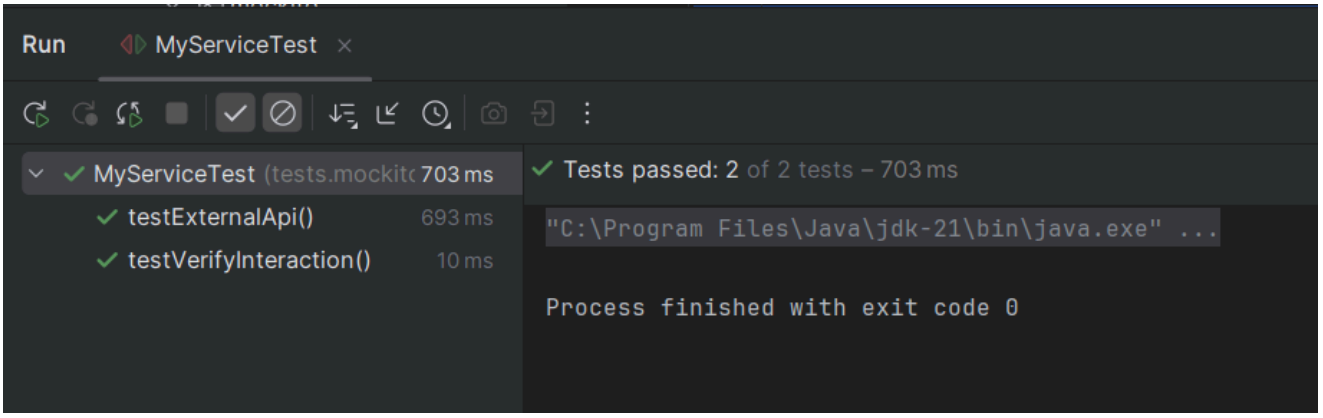
Here in this case we can see that as the `fetchData()` method is already called once, `verify(mockApi).getData()` will be successful.

## Code:

```
package tests.mockito;

import exercises.mokito.ExternalApi;
import exercises.mokito.MyService;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;

import static org.mockito.Mockito.*;

public class MyServiceTest {
    @Test
    public void testExternalApi() {
        ExternalApi mockApi = Mockito.mock(ExternalApi.class);
        when(mockApi.getData()).thenReturn("Mock Data");
```

```
        MyService service = new MyService(mockApi);
        String result = service.fetchData();
        Assertions.assertEquals("Mock Data", result);
    }

    @Test
    public void testVerifyInteraction() {
        ExternalApi mockApi = Mockito.mock(ExternalApi.class);
        MyService service = new MyService(mockApi);
        service.fetchData();
        verify(mockApi).getData();
    }
}
```

## Results:



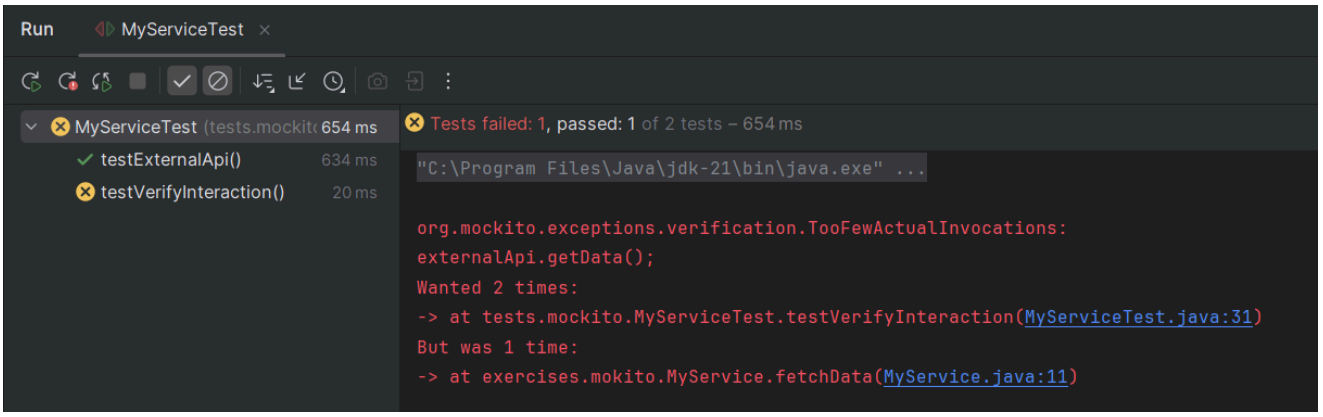

# Type-2

Here in this part of code, we tried to verify that `getData()` method is called minimum of 2 times, but as that method is invoked only once, this verification is failed.

## Code:

```
@Test
public void testVerifyInteraction() {
    ExternalApi mockApi = Mockito.mock(ExternalApi.class);
    MyService service = new MyService(mockApi);
    service.fetchData();
    verify(mockApi, times(2)).getData();
}
```

## Results:




---

# Other exercises

## Exercise 3: Argument Matching

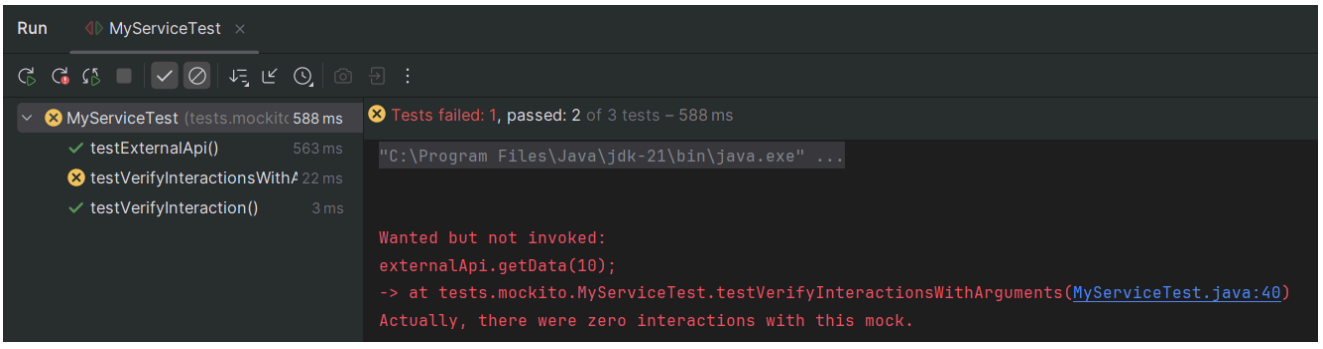
### Type-1

Here we can see that as the `getData()` method with `same arguments` or `not at all` invoked, so it would throw an error as `method is not invoked` during verification.

### Code:

```
@Test
public void testVerifyInteractionsWithArguments(){
    ExternalApi mockApi = Mockito.mock(ExternalApi.class);
    MyService service = new MyService(mockApi);
    // Verify with arguments
     verify(mockApi).getData(10);
}
```

### Results:




### Type-2:

Here as the `getData()` method with same arguments i.e., 10 is invoked, this verification will be successful.

## Code:

```
@Test
public void testVerifyInteractionsWithArguments(){
    ExternalApi mockApi = Mockito.mock(ExternalApi.class);
    MyService service = new MyService(mockApi);
    service.fetchData(10);
    // Verify with arguments
     verify(mockApi).getData(10);
}
```

## Results