

THIAGO FREIRE DE CARVALHO

ATIVIDADE 1

TESTES DE SOFTWARE

Problema escolhido via Stack Overflow

Através da string de busca “[unit-testing] or [junit] or [pytest]” foi escolhida a seguinte pergunta abaixo:

How to mock void methods with Mockito

Asked 14 years, 5 months ago Modified 2 years, 6 months ago Viewed 1.3m times



How to mock methods with void return type?

1213

I implemented an observer pattern but I can't mock it with Mockito because I don't know how.



And I tried to find an example on the Internet but didn't succeed.

My class looks like this:



```
public class World {

    List<Listener> listeners;

    void addListener(Listener item) {
        listeners.add(item);
    }

    void doAction(Action goal, Object obj) {
        setState("i received");
        goal.doAction(obj);
        setState("i finished");
    }

    private String state;
    //setter getter state
}

public class WorldTest implements Listener {

    @Test public void word{
        World w= mock(World.class);
        w.addListener(this);
        ...
        ...
    }
}

interface Listener {
    void doAction();
}
```

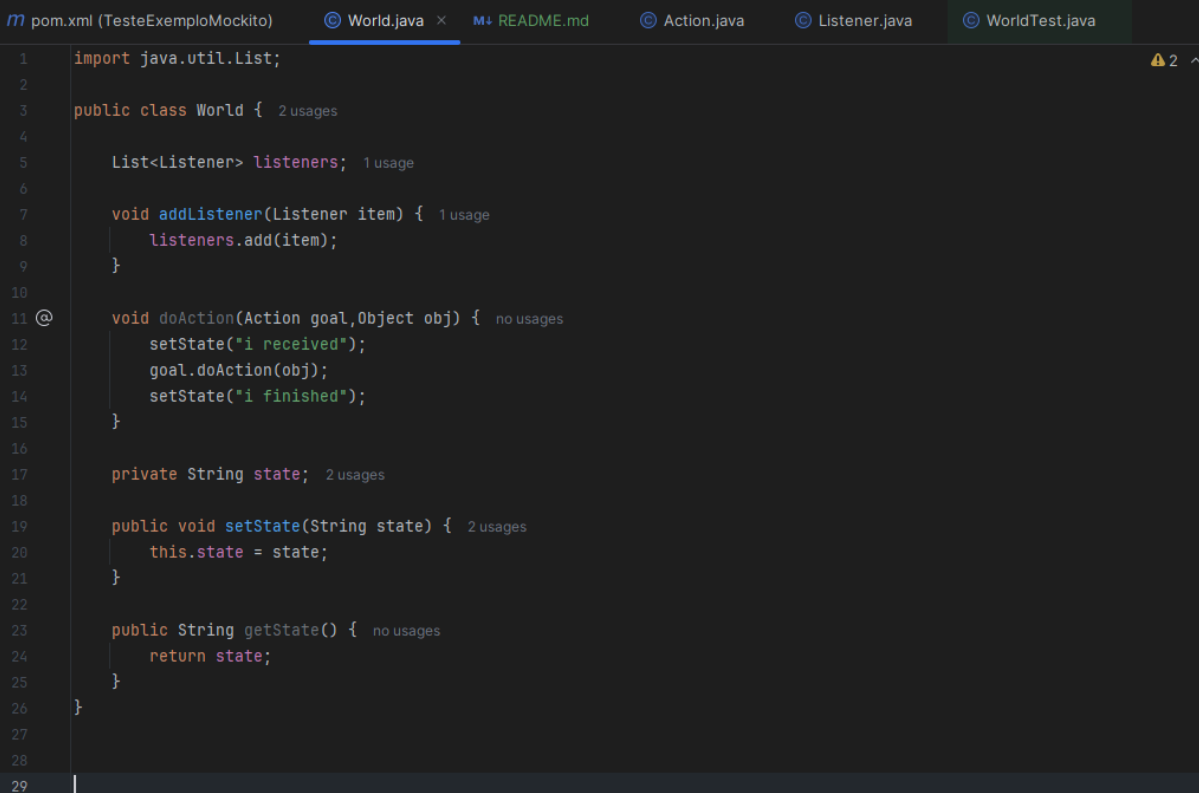
The system is not triggered with mock.

Na pergunta em questão, o desenvolvedor se questiona de que forma pode testar e mockar um método de uma classe cujo retorno é *void*. Para demonstrar o problema, o usuário fornece uma replicação incompleta do código fonte, apenas para fim de entendimento da dúvida.

Reprodução do problema

Em princípio, vale ressaltar que, devido à incompletude do código fornecido, algumas classes e métodos foram implementados de forma que possam não corresponder com a realidade do código original. No entanto, foram necessários para reprodução do problema, e não afetam a demonstração deste.

Foi escolhida a IDE IntelliJ IDEA, devido ao seu foco em desenvolvimento em projetos Java.



```
m pom.xml (TesteExemploMockito)  World.java x  README.md  Action.java  Listener.java  WorldTest.java
1  import java.util.List;
2
3  public class World { 2 usages
4
5      List<Listener> listeners; 1 usage
6
7      void addListener(Listener item) { 1 usage
8          listeners.add(item);
9      }
10
11  @ void doAction(Action goal, Object obj) { no usages
12      setState("i received");
13      goal.doAction(obj);
14      setState("i finished");
15  }
16
17  private String state; 2 usages
18
19  public void setState(String state) { 2 usages
20      this.state = state;
21  }
22
23  public String getState() { no usages
24      return state;
25  }
26 }
27
28
29
```

```
m pom.xml (TesteExemploMockito)  World.java  README.md  Action.java  Listener.java  WorldTest.java x
1 import static org.junit.jupiter.api.Assertions.*;
2 import static org.mockito.Mockito.*;
3
4
5
6 class WorldTest { no usages
7     public void testWorld(){ no usages
8         World w = mock(World.class);
9         Listener listener = new Listener();
10        w.addListener(listener);
11    }
12 }
13
14
15
```

Disponível no [Repositório do Github](#).

Problema

O usuário deseja utilizar a biblioteca Mockito para aplicar o mock em métodos com retorno do tipo *void* da classe *World*, a fim de realizar testes relacionados aos seus comportamentos.

Resolução

Na resposta aceita pelo Stack Overflow, o usuário recomenda utilizar os métodos feitos para testes em métodos *void* de acordo com a recomendação da documentação do próprio Mockito.

11 Answers

Sorted by: Highest score (default) ▾



1478

Take a look at the Mockito [API docs](#). As the linked document mentions (Point # 12) you can use any of the `doThrow()`, `doAnswer()`, `doNothing()`, `doReturn()` family of methods from Mockito framework to mock void methods.



For example,



```
Mockito.doThrow(new Exception()).when(instance).methodName();
```



or if you want to combine it with follow-up behavior,

+200



```
Mockito.doThrow(new Exception()).doNothing().when(instance).methodName();
```

De acordo com o [link](#) enviado pelo usuário, é possível ver que a documentação da API do Mockito informa o caso de uso dos seguintes métodos:

You can use `doThrow()`, `doAnswer()`, `doNothing()`, `doReturn()` and `doCallRealMethod()` in place of the corresponding call with `when()`, for any method. It is necessary when you

- *stub void methods*
- *stub methods on spy objects (see below)*
- *stub the same method more than once, to change the behaviour of a mock in the middle of a test*

O usuário ainda disponibiliza um trecho de código utilizando o método `doAnswer`.

Presuming that you are looking at mocking the setter `setState(String s)` in the class `World` below is the code uses `doAnswer` method to mock the `setState`.

```
World mockWorld = mock(World.class); 1
doAnswer(new Answer<Void>() {
    public Void answer(InvocationOnMock invocation) {
        Object[] args = invocation.getArguments();
        System.out.println("called with arguments: " + Arrays.toString(args));
        return null;
    }
}).when(mockWorld).setState(anyString()); 2
```

Este método permite ao programador uma maior observabilidade e manipulação do comportamento do método. No exemplo acima, ele o utiliza para checar os argumentos que foram enviados para o método `setState` da classe `World`.

Em 1 (na imagem) ele efetua o mock da classe em uma instância.

Já em 2 (na imagem), é feito o mock no método `setState` para `anyString`. Isso significa que, para qualquer chamada posterior a essa linha, sempre que o `setState` for chamado para qualquer valor de string, o comportamento dentro da função `doAnswer` será executado. No caso acima, os argumentos enviados para o método são impressos na saída padrão.

Na IDE, isso pode ser replicado da seguinte forma:

```
class WorldTest {  
    @Test  
    public void testWorld(){  
        World mockWorld = mock(World.class);  
        doAnswer(invocation -> {  
            Object[] args = invocation.getArguments();  
            System.out.println("called with arguments: " + Arrays.toString(args));  
            return null;  
        }).when(mockWorld).setState(anyString());  
  
        mockWorld.setState("teste");  
    }  
}
```

Note que, adicionalmente ao exemplo fornecido pelo o usuário no StackOverflow, foi necessário incluir a chamada ao método a partir da instância da classe com o mock, isso é o que vai permitir testarmos de fato a execução.

Após a execução, o teste passa com sucesso e é exibido o log mostrando que o método foi chamado com apenas um argumento de valor “teste”.

```
✓ Tests passed: 1 of 1 test - 1sec 86ms  
C:\Users\Thiago\.jdk\corretto-17.0.11\bin\java.exe ...  
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended  
called with arguments: [teste]  
  
Process finished with exit code 0
```

Poderíamos também testar se a execução do método está sendo feita corretamente através do número de argumentos enviados:

```
@Test  
public void testWorldNumberOfArguments(){  
    World mockWorld = mock(World.class);  
    doAnswer(invocation -> {  
        Object[] args = invocation.getArguments();  
        System.out.println("Called with number of arguments " + args.length);  
        assert(args.length == 1);  
        return null;  
    }).when(mockWorld).setState(anyString());  
  
    mockWorld.setState("teste");  
}
```

No exemplo acima, é feito um teste para garantir que a quantidade de argumentos enviados ao método é igual a 1. Em seguida, chamamos o método com 1 argumento e rodamos o teste:

```
✓ Tests passed: 1 of 1 test - 1sec 87ms
C:\Users\Thiago\.jdk\corretto-17.0.11\bin\java.exe ...
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
Called with number of arguments 1

Process finished with exit code 0
```

O teste é executado com sucesso e exibe a informação do número de argumentos.

O código fonte com a replicação do exemplo está disponível no [Repositório do Github](#).

Respostas não aceitas

Existem vários motivos para a não aceitação de uma resposta no StackOverflow. Algumas vezes, a resposta em si não contém erros, porém não é aceita pela avaliação dos usuários, seja porque não responde de maneira objetiva, seja por não utilizar exemplos, ou simplesmente por serem registradas anos depois. No caso da pergunta em questão, as outras respostas não foram aceitas pois fogem da intenção original da pergunta, já que o usuário deseja de fato aplicar um mock ao método do tipo *void*, e as outras respostas recomendam que ele utilize o Mockito para chamar o método real ou simplesmente ignorá los com o uso da função *spy*.

Respostas não aceitas - Os usuários recomendam que seja utilizado o *doCallRealMethod*. Esse método não faz o mock do comportamento, apenas executa o método real diretamente.

▲ I think I've found a simpler answer to that question, to call the real method for just one method (even if it has a void return) you can do this:

138


▼

```
Mockito.doCallRealMethod().when(<objectInstance>).<method>();
<objectInstance>.<method>();
```



Or, you could call the real method for all methods of that class, doing this:

```
<Object> <objectInstance> = mock(<Object>.class, Mockito.CALLS_REAL_METHODS);
```



Adding another answer to the bunch (no pun intended)...

18

You do need to call the `doAnswer` method if you can't/don't want to use spy's. However, you don't necessarily need to roll your own [Answer](#). There are several default implementations. Notably, [CallsRealMethods](#).

In practice, it looks something like this:


```
doAnswer(new CallsRealMethods()).when(mock)
    .voidMethod(any(SomeParamClass.class));
```

Or:


```
doAnswer(Answers.CALLS_REAL_METHODS.get()).when(mock)
    .voidMethod(any(SomeParamClass.class));
```

Share Follow

edited Mar 31, 2014 at 12:31


**poussma**
7,223 ● 3 ● 47 ● 69

answered Mar 19, 2014 at 17:41

**javamonkey79**
17.7k ● 38 ● 115 ● 177

Add a comment

Respostas não aceitas - Usuários recomendam que se utilize o *spy* para prevenir que o teste chame os métodos *void*. Essa não é a intenção original do usuário.




Adding to what @sateesh said, when you just want to mock a void method in order to prevent the test from calling it, you could use a `Spy` this way:

119

```
World world = new World();
World spy = Mockito.spy(world);
Mockito.doNothing().when(spy).methodToMock();
```

When you want to run your test, make sure you call the method in test on the `spy` object and not on the `world` object. For example:

```
assertEquals(0, spy.methodToTestThatShouldReturnZero());
```



First of all: you should always import mockito static, this way the code will be much more readable (and intuitive):

49

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

For partial mocking and still keeping original functionality on the rest mockito offers "Spy".

You can use it as follows:

```
private World world = spy(new World());
```


Respostas não aceitas - Os usuários recomendam o mesmo uso do *doAnswer*, porém foi registrada anos depois, e já existia uma resposta selecionada pelo StackOverflow:



35



How to mock void methods with mockito - there are two options:

1. `doAnswer` - If we want our mocked void method to do something (mock the behavior despite being void).
2. `doThrow` - Then there is `Mockito.doThrow()` if you want to throw an exception from the mocked void method.

Following is an example of how to use it (not an ideal usecase but just wanted to illustrate the basic usage).

```
@Test
public void testUpdate() {

    doAnswer(new Answer<Void>() {

        @Override
        public Void answer(InvocationOnMock invocation) throws Throwable {
            Object[] arguments = invocation.getArguments();
            if (arguments != null && arguments.length > 1 && arguments[0] != null

                Customer customer = (Customer) arguments[0];
                String email = (String) arguments[1];
                customer.setEmail(email);

            }
            return null;
        }
    }).when(daoMock).updateEmail(any(Customer.class), any(String.class));

    // calling the method under test
    Customer customer = service.changeEmail("old@test.com", "new@test.com");

    //some asserts
    assertThat(customer, is(notNullValue()));
    assertThat(customer.getEmail(), is(equalTo("new@test.com")));
}

@Test(expected = RuntimeException.class)
public void testUpdate_throwsException() {

    doThrow(RuntimeException.class).when(daoMock).updateEmail(any(Customer.class),

    // calling the method under test
```

You could find more details on how to **mock** and **test void** methods with Mockito in my post [How to mock with Mockito \(A comprehensive guide with examples\)](#)



If you need to do some operations in the mocked void method, and you need to manipulate the argument that sent to void method; you can combine Mockito.doAnswer with

2

ArgumentCaptor.capture method.



Let's say you have **SpaceService** that autowires a **GalaxyService**, which has a void method called **someServiceMethod**.



You want to write test for one of your method in **SpaceService** that calls **GalaxyService**'s void method. Your planet is also generated inside SpaceService. So you don't have any chance to mock that.

Here is your sample **SpaceService** class that you want to write tests for.