



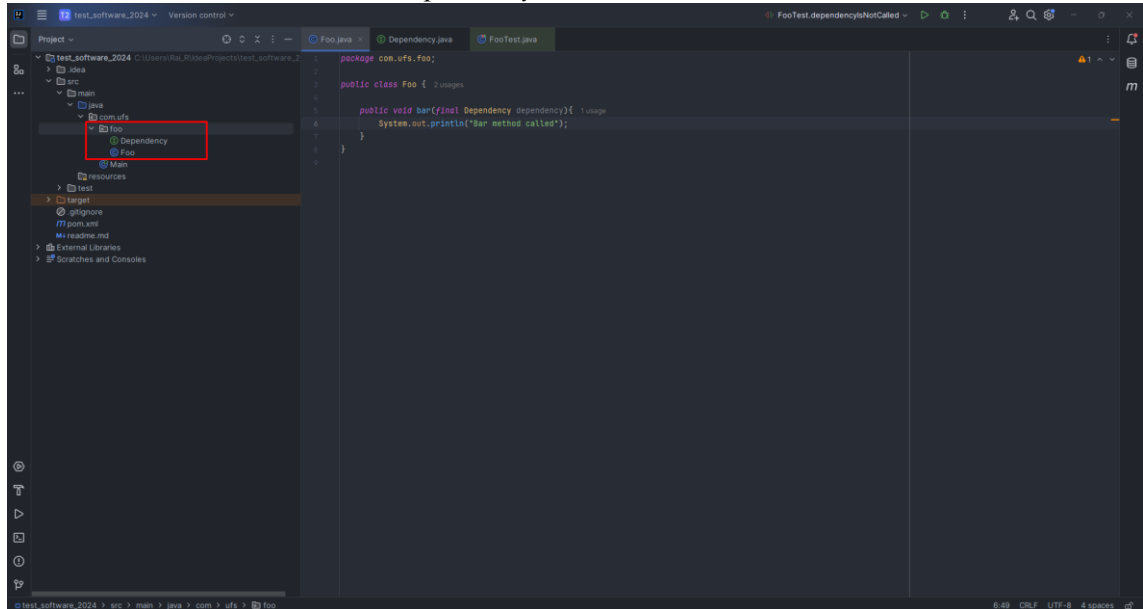
RAI RAFAEL SANTOS SILVA
ATIVIDADE 1 – TESTES UNITÁRIOS E O STACK
OVERFLOW

COMP0444 - TESTE DE SOFTWARE
GLAUCO DE FIGUEIREDO CARNEIRO

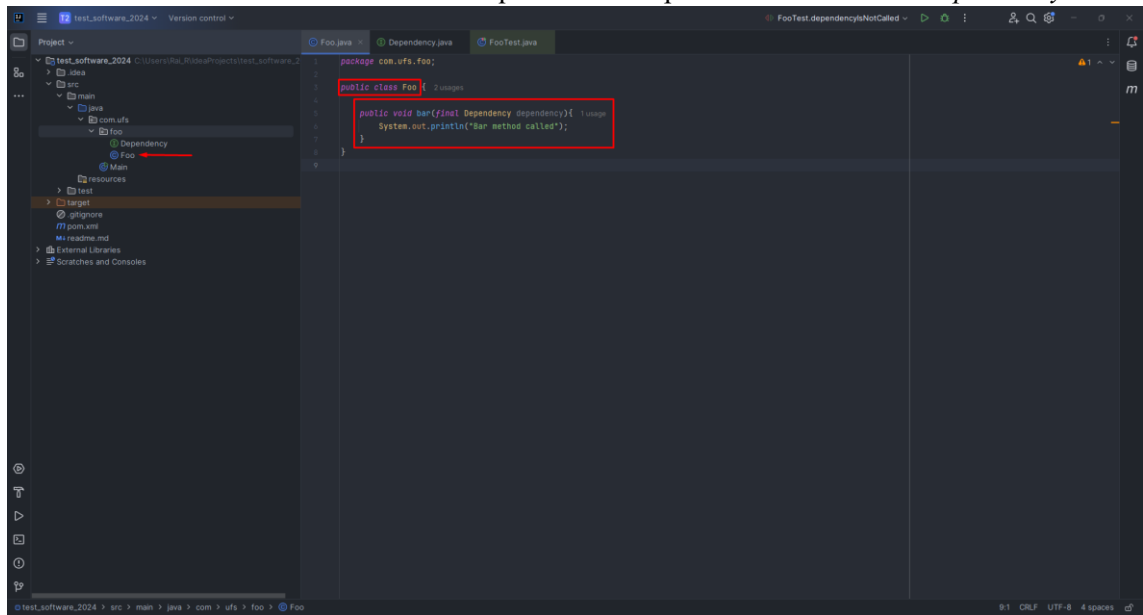
1 Desenvolvimento da Atividade

1.1 Implementação do código

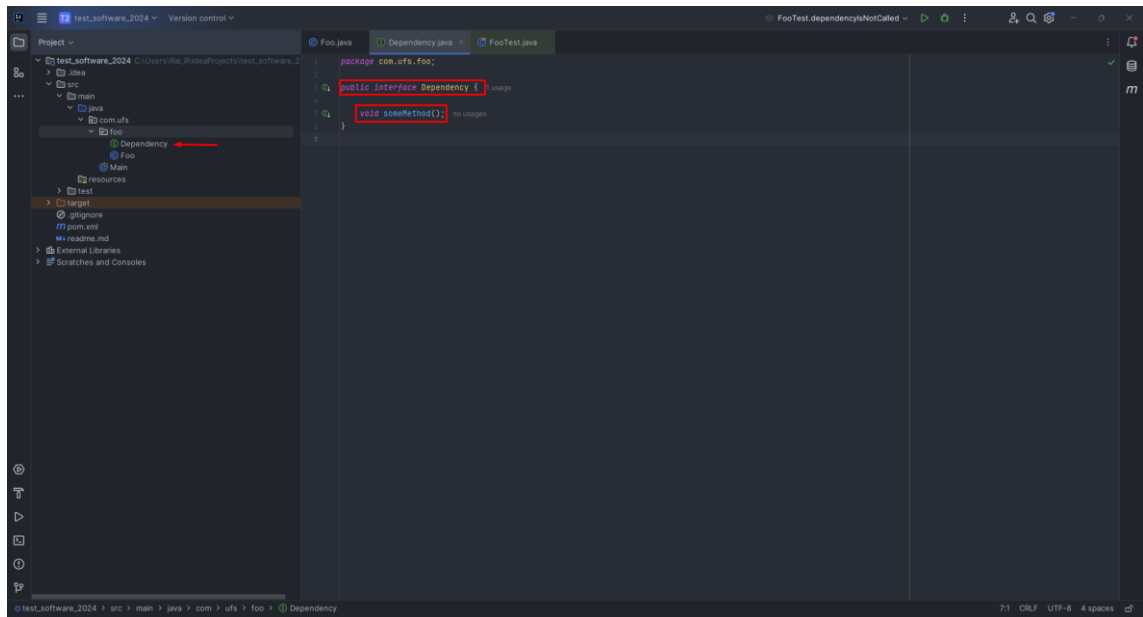
No primeiro momento, foi realizada a criação de dois arquivos dentro do pacote *foo*. Uma classe *Foo* e uma interface *Dependency*.



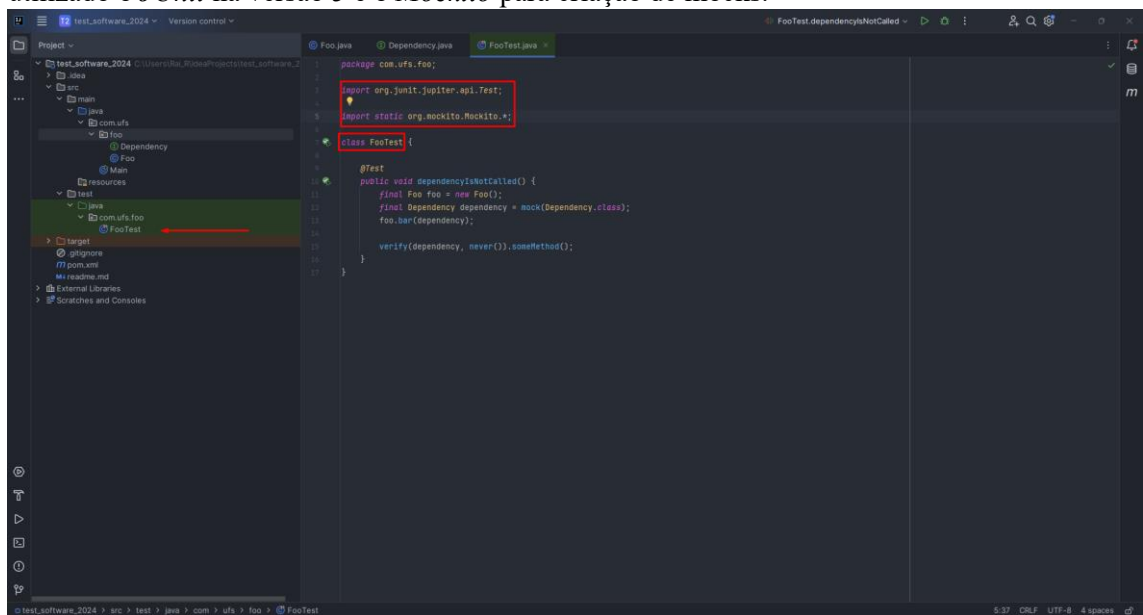
Na classe *Foo* foi criado um método *bar* que tem como parâmetro a interface *Dependency*.



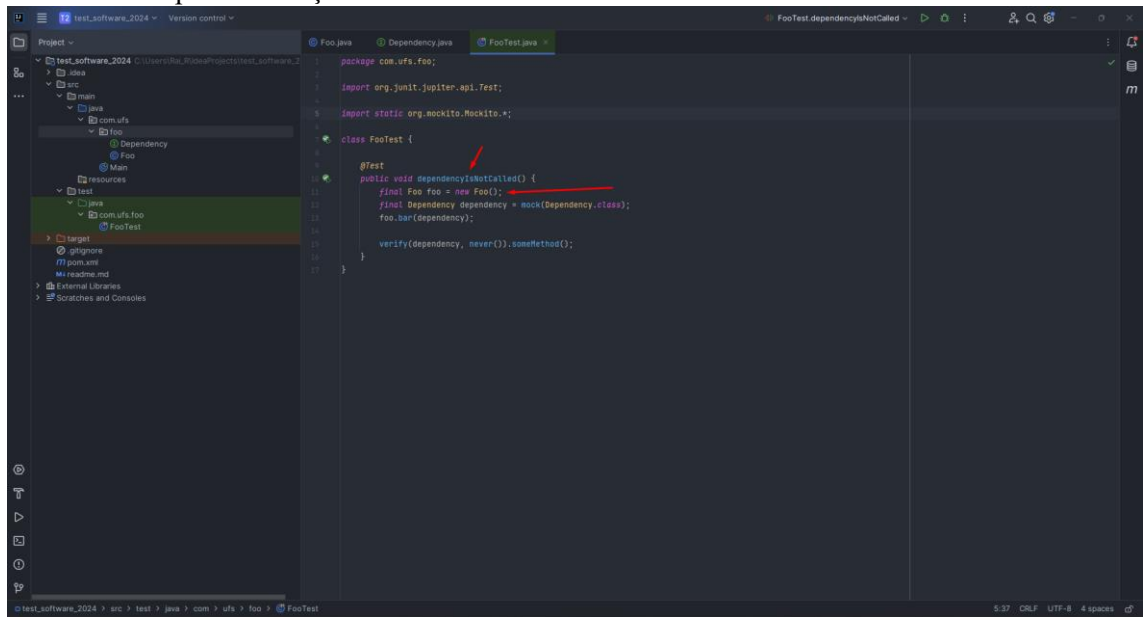
Na interface *Dependency*, que simula uma dependência de uma classe (a classe *Foo*), foi adicionado a escritura de um método chamado *someMethod*.



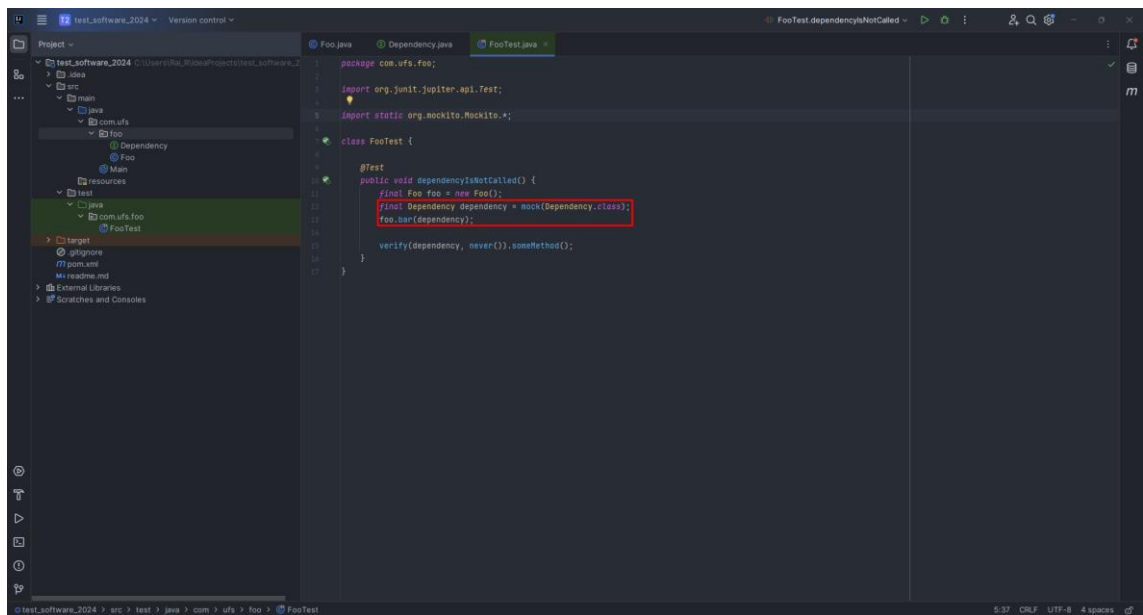
Então foi criada uma classe de teste para a classe *Foo*. Para a realização do teste está sendo utilizado o *JUnit* na versão 5 e o *Mockito* para criação de mocks.



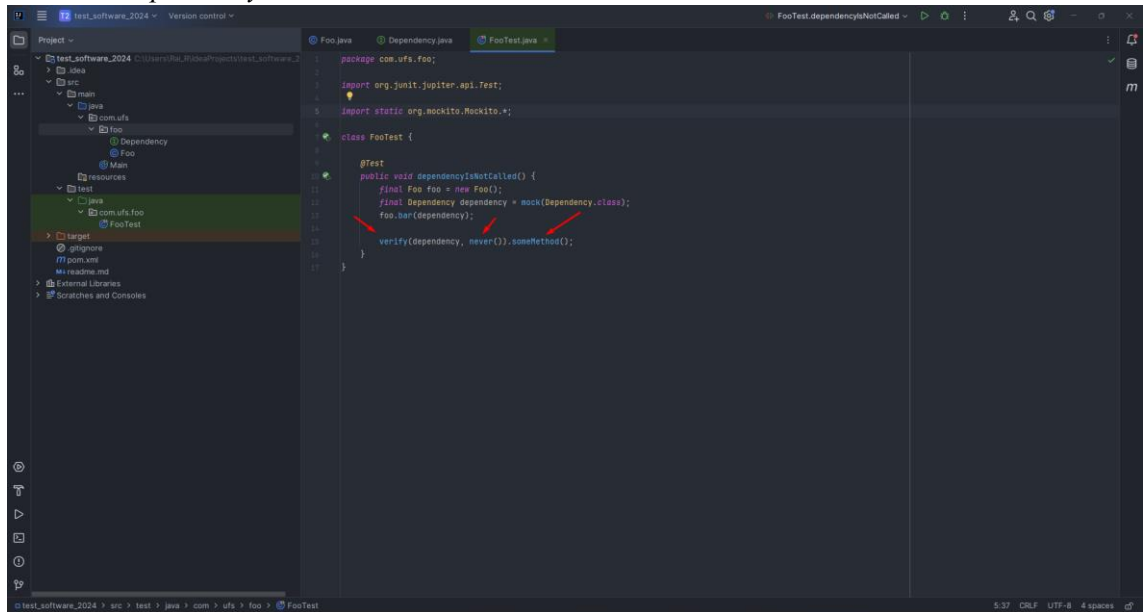
Em seguida foi criado um teste e anotado com a *annotation* `@Test` e foi instanciado um objeto da classe `Foo` para realização de testes na classe.



Então foi gerado um mock da interface `Dependency` e o mock gerado foi injetado no método `Bar` da classe `Foo`.

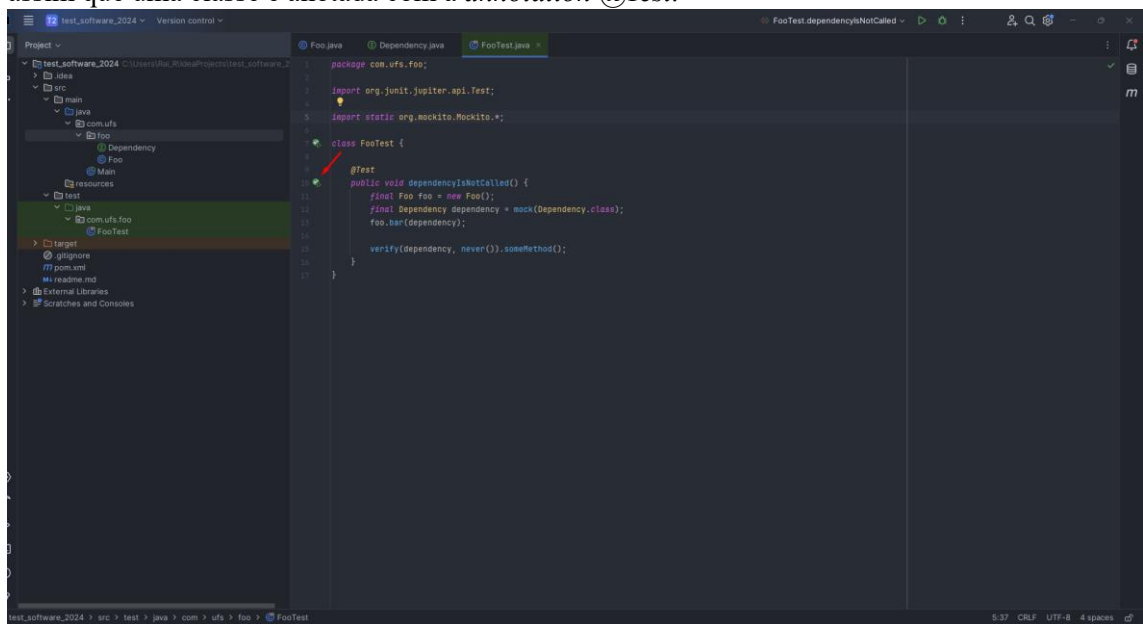


Por último foi utilizado o método `verify` do Mockito, passando como argumento a dependência injetada e o método `never()` para averiguar que o método `someMethod()` da interface `Dependency` não foi invocado.

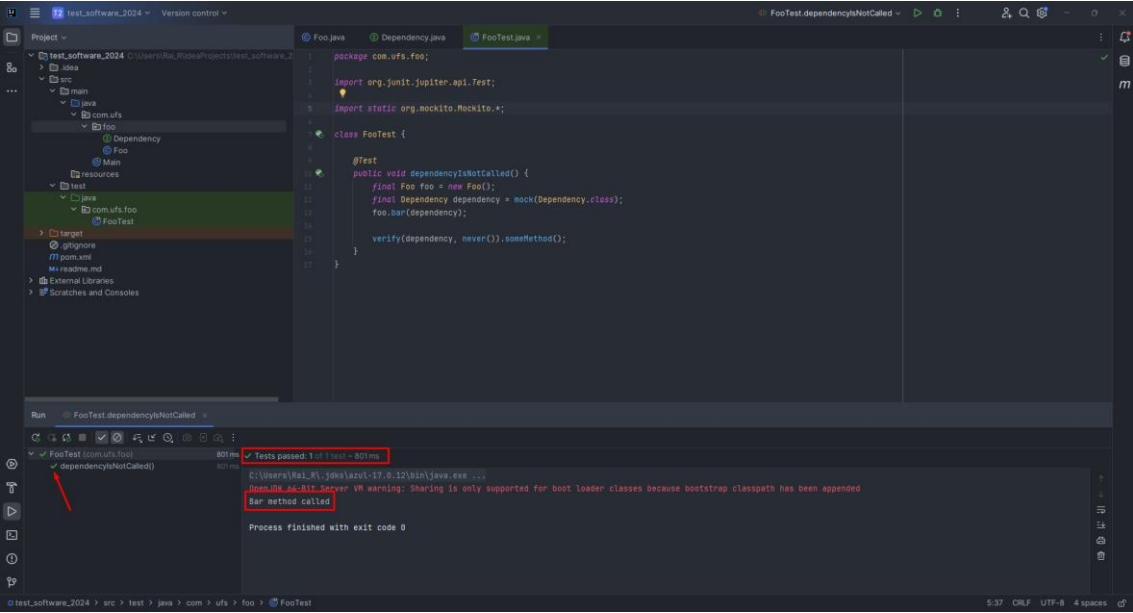


1.2 Execução dos Testes

Para execução dos testes, a própria IDE (IntelliJ) oferece a funcionalidade de executar o teste assim que uma classe é anotada com a *annotation* `@Test`.



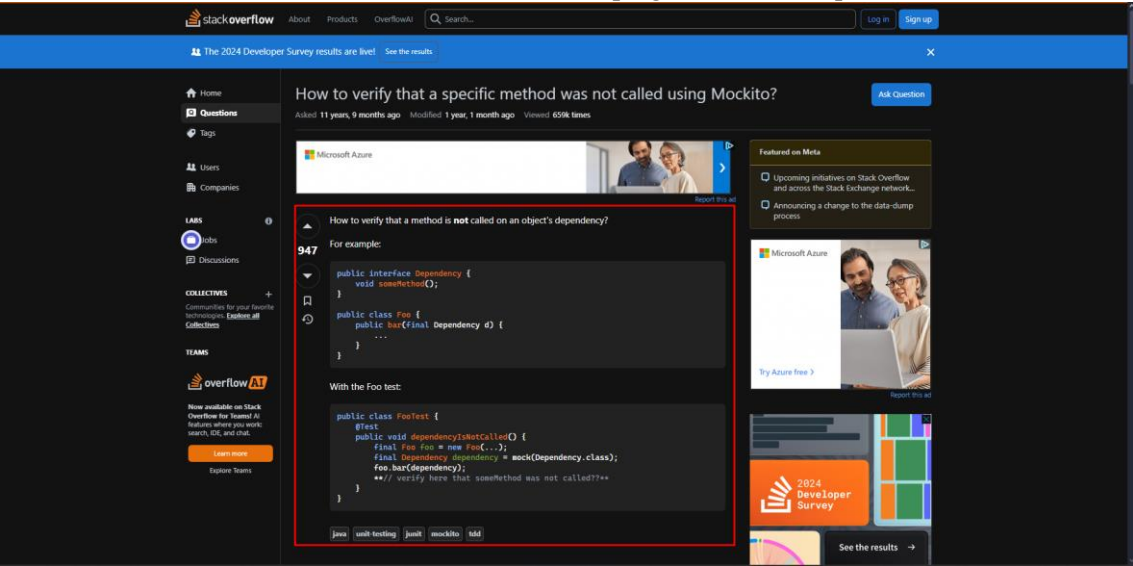
Ao executar o teste em questão, é executado a linha dentro do método bar que imprime “Bar method called” e em seguida o JUnit verifica que não há chamada de um método específico (no caso o método *someMethod()*) da interface *Dependency*.



2 Outras respostas

2.1 Resposta principal

Ao acessar o [Stack Overflow](#), vemos inicialmente a pergunta realizada pelo usuário.



Em seguida vemos que para esta pergunta foi respondida com 7 respostas e que a resposta principal teve a pontuação de 1550, pois foi clara e concisa ao responder o usuário.

stackoverflow

7 Answers

Sorted by: Highest score (default)

1550

Even more meaningful:

```
import static org.mockito.Mockito.never;
import static org.mockito.Mockito.verify;

// ...

verify(dependency, never()).someMethod();
```

The documentation of this feature is there [\[4\]](#) "Verifying exact number of invocations / at least x / never", and the `never` javadoc is [here](#).

Share Improve this answer Follow

edited Jul 16, 2017 at 6:26 answered Oct 12, 2012 at 16:08

Michael Xin Sun 1,224 ● 11 ● 12

100 Using `never` is the best and most specific way, but if you need to check an entire mock object, also consider `verifyZeroInteractions(mockObject)` or `verifyNoMoreInteractions(mockObject)`. – [ari rowman](#) Oct 12, 2012 at 19:18

1 what to do if someMethod is private? – [Sumit Kumar Saha](#) Aug 1, 2019 at 13:43

1 Then you can not mock it in the first place (with Mockito) ; PowerMock allows to that but it's more complex to set up. Or if you have ownership of the code, you relax the visibility to package. – [brnc3](#) Aug 1, 2019 at 14:29

2 Since 3.0.1 `verifyZeroInteractions` has been deprecated. `verifyNoInteractions` is the suggested alternative. Mockito version at the time of this comment is 3.3.3 – [Vik](#) Apr 29, 2020 at 17:51

@brnc3 generally private methods shouldn't be tested, but if you do want to test them, that can be a good sign that the method should be more visible, potentially even extracted to a separate class altogether. – [Hugo](#) Apr 27, 2023 at 5:02

Show 1 more comment

Microsoft Azure

Hot Network Questions

2.2 Outras perguntas

Ao rolar a página, encontramos mais uma resposta que também responde à pergunta do usuário, mas não de forma tão clara quanto a pergunta mais votada. Portanto essa resposta recebeu menos pontuação.

stackoverflow

178

Use the second argument on the `Mockito.verify` method, as in:

```
Mockito.verify(dependency, Mockito.times(0)).someMethod();
```

Share Improve this answer Follow

edited Jan 11, 2021 at 17:15 answered Oct 12, 2012 at 15:44

Naman 38.6k ● 29 ● 229 ● 365

16 public static VerificationMode never() { return times(0); } – [gbrp](#) Apr 25, 2017 at 12:20

12 `never()` is not significantly more readable than `times(0)`. But the existence of `never` does increase cognitive load and makes the Mockito system harder to understand and remember how to use. So really Mockito shouldn't have included `never` in their API, its not worth the mental cost. – [B T](#) Jan 14, 2019 at 21:56

1 Question: does this form verify that `someMethod` was called 0 times, or does it only verify that `someMethod` was never called with zero arguments? – [B T](#) Jan 14, 2019 at 21:57

1 @B T - I would imagine it verifies the `someMethod` with zero arguments was called zero times - not verified. – [Johannes](#) Jan 13, 2019 at 9:25

Same works for Mockito `times(0)` – [fakrudenn](#) Nov 12, 2021 at 18:47

Add a comment

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

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

There are actually many ways to achieve this, however it's (arguably) clearer to use the

```
verify(yourMock, times(0)).someMethod();
```

method all over your tests, when on other tests you use it to assert a certain amount of executions like this:

Hot Network Questions

Temos mais abaixo na mesma página outras respostas que também resolvem o problema, mas que receberam menos pontuação ainda, pois está ainda menos clara e concisa ao responder a resposta principal.

