JULIO GEOVANI OLIVEIRA GUIMARAES – ATIVIDADE 1

Problema: How to verify a method is called two times with mockito verify()?

```
org.mockito.exceptions.verification.TooManyActualInvocations:
Wanted 1 time:
But was 2 times. Undesired invocation:
```

Descrição do problema: O programador está tendo o problema em que um determinado método está sendo invocado uma quantidade de vezes indesejada. Ele deseja saber como verificar se um método é chamado duas vezes com mockito verify().

Solução Aprovada: Para solucionar este problema, basta utilizar o VerificationMode de forma correta e apropriada para os testes em específicos:

```
import static org.mockito.Mockito.atLeast;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;

verify(mockObject, atLeast(2)).someMethod("was called at least twice");
verify(mockObject, times(3)).someMethod("was called exactly three times");
```

Solução que abordei na minha IDE:

1 - Criar método de multiplicação para dois atributos.

```
1  public class Multiplier {
2      public int multiply(int a, int b) {
3          return a * b;
4      }
5  }
```

2 – Implementar classe e método para realização da multiplicação.

```java
public class MathService {
    private Multiplier multiplier;

    public void setMultiplier(Multiplier multiplier) {
        this.multiplier = multiplier;
    }

    public int multiply(int a, int b) {
        return multiplier.multiply(a, b);
    }
}
```

3 - Importar os métodos utilizados na solução do problema:

```java
import static org.mockito.Mockito.atLeast;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
```

atLeast - Verifica se o método foi chamado pelo menos uma quantidade específica de vezes.
times - Verifica se um método foi chamado exatamente um número específico de vezes.
verify - Verifica se um método foi chamado no mock.

4 - Implementar classes e métodos para realizar os testes em específicos.

```java
import static org.mockito.Mockito.atLeast;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import org.mockito.Mockito;
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;

public class MathServiceTest {

    private MathService mathService;
    private Multiplier multiplierMock;

    @Before
    public void setUp() {
        mathService = new MathService();
        multiplierMock = Mockito.mock(Multiplier.class);
        mathService.setMultiplier(multiplierMock);
    }

    @Test
    public void testMultiplyCalledAtLeastTwice() {

        when(multiplierMock.multiply(10, 10)).thenReturn(100);


        mathService.multiply(10, 10);
        mathService.multiply(10, 10);

        // Verifica se o método multiply do mock foi chamado pelo menos duas vezes!
        verify(multiplierMock, atLeast(2)).multiply(10, 10);

        // Verifica se o método multiply do mock foi chamado exatamente três vezes!
        verify(multiplierMock, times(3)).multiply(10,10);
    }
}
```

Por fim, temos o resultado dos testes. A utilização do atLeast verifica se o método foi testado pelo menos duas vezes, mas não retorna falha caso tenham discrepâncias na quantidade de testes. Essa solução foi aprovada, pois, é uma solução mais completa e que abrange menos possíveis problemas de acordo com os testes realizados.

Solução não aprovada: Utilizar o verify apenas com o método “Times”.

Código:

```
// MyCallback.java
public interface MyCallback {
  void someMethod(String value);
}
// MyTestableManager.java
public class MyTestableManager {
  private MyCallback callback;

  public MyTestableManager(MyCallback callback) {
    this.callback = callback;
  }

  public void perform(){
    callback.someMethod("first");
    callback.someMethod("second");
    callback.someMethod("third");
  }
}
```

Teste com verify e times:

```
import org.mockito.Mockito.times;
import org.mockito.Mockito.verify;
import org.mockito.Mock;
import org.mockito.Captor;
// whatever other imports you need
@Mock
private MyCallback callback;
@Captor
private ArgumentCaptor<String> captor;

private MyTestableManager uut = new MyTestableManager(callback);

// in your test method:
uut.perform()

verify(callback, times(3)).someMethod(captor.capture())

assertTrue(captor.getAllValues().get(0) == "first")
assertTrue(captor.getAllValues().get(1) == "second")
assertTrue(captor.getAllValues().get(2) == "third")
```

Principal problema:

A utilização apenas do times(3) exige que o método seja chamado exatamente três vezes. Caso o método seja chamado uma quantidade de vezes inferior ou superior a 3 vezes, o teste falhará. Nesse caso, quando o número de chamadas do método é variável, o ideal é sempre utilizar o atLeast(3). Por isso, esta solução não foi aprovada.

GitHub: https://github.com/JulioDEV11/Teste_Software_2024_GUIMARAES_JULIO