UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CORNÉLIO PROCÓPIO

Mocking – Funcionalidades adicionais

Este material foi elaborado com base na utilização dos frameworks JUnit e Mockito.

Veja os códigos utilizados em:

https://github.com/andreendo/software-testing-undergrad-course/tree/master/mockingExtra

- **Como eu faço para "mockar" um método que modifica um parâmetro?**

Estamos acostumados criar mocks para métodos que retornam algum valor ou objeto; nesse caso, usamos a estrutura *when(___).thenReturn(___)*. No entanto, às vezes é necessário mockar um método no qual a parte que queremos controlar não é o retorno e sim um objeto que foi passado como parâmetro e modificado. Considere a classe a seguir.

```
public class Avaliador {
    private BD bd;

    public void setBd(BD bd) {
        this.bd = bd;
    }

    public boolean ehClienteArriscado(Cliente cliente) {
        //objeto cliente pode conter apenas o id
        //este metodo completa os demais atrbitutos com dados do BD
        bd.completarDados(cliente);

        if(cliente.getRisco() >= 10)
            return true;

        return false;
    }
}
```

Neste exemplo hipotético, você gostaria de mockar o método completarDados() e configurar o risco para o valor exato de 10. No Mockito, podemos usar a estrutura doAnswer(). Usando essa estrutura podemos acessar os parâmetros do método que queremos mockar e alterar seus valores. Nosso problema específico é tratado no trecho de código apresentado a seguir.

```
public class AvaliadorTest {

    @Test
    public void clienteComRisco10Test() {
        BD bdMock = mock(BD.class);

        Avaliador avaliador = new Avaliador();
        avaliador.setBd(bdMock);

        Cliente cliente = new Cliente();
        cliente.setId(55667);

        //aqui acontece a magica
        doAnswer(new Answer() {     //cria uma classe anonima
            @Override
            public Object answer(InvocationOnMock invocation) throws Throwable {
                //recuperar o primeiro argumento do metodo
                //e alterar seu risco para 10
                Cliente c = invocation.getArgumentAt(0, Cliente.class);
                c.setRisco(10);
                return null;    //eh um metodo void
            }
        }).when(bdMock).completarDados(cliente);

        assertTrue( avaliador.ehClienteArriscado(cliente) );
    }
}
```

- **Como eu faço para “mockar” apenas um método enquanto que os outros métodos usam a implementação original?**

Quando usamos o método “mock(__)”, todos os métodos da classe ou interface precisam ser mockados. No entanto, em alguns casos queremos mockar apenas um método e usar o código original dos outros métodos. Essa situação é chamada de *mock parcial*. Vejamos o exemplo a seguir (baseado em [Mokkapaty01]).

```
public class Empregado {
    private long id;
    private String primeiroNome;
    private String sobrenome;

    public Empregado(long id, String primeiroNome, String sobrenome) {
        this.id = id;
        this.primeiroNome = primeiroNome;
        this.sobrenome = sobrenome;
    }

    public String getNomeCompleto() {
        return getPrimeiroNome() + " " + getSobrenome();
    }

    public long getId() {
        return id;
```

Neste exemplo, queremos testar o método “getNomeCompleto()” (usar sua implementação original) e mockar apenas os métodos “getPrimeiroNome()” e “getSobrenome()”. No Mockito, temos uma estrutura chamada de ***spy (espiar)***. Nosso problema específico é tratado no trecho de código apresentado a seguir.

```
public class EmpregadoTest {

    @Test
    public void getNomeCompletoTest() {
        //criar o objeto
        Empregado empregado = new Empregado();
        //especifica que o objeto sera "espionado" via o spyEmpregado
        Empregado spyEmpregado = spy(empregado);

        //apenas esses dois metodos serao mockados
        when(spyEmpregado.getPrimeiroNome()).thenReturn("Maria");
        when(spyEmpregado.getSobrenome()).thenReturn("Lacrosse");

        assertEquals("Maria Lacrosse", spyEmpregado.getNomeCompleto());

        //verifica se os metodos foram chamados na ordem correta
        InOrder inOrder = inOrder(spyEmpregado);
        inOrder.verify(spyEmpregado).getPrimeiroNome();
        inOrder.verify(spyEmpregado).getSobrenome();
    }
}
```

Usamos o método estático spy() ao invés de mock() para resolver esse problema. As demais estruturas seguem a sintaxe dos mocks tradicionais.

- **Como eu faço para “mockar” um método estático?**

O Mockito possui a limitação de não ser capaz de mockar métodos estáticos. Caso você precise mockar um método estático, é necessário utilizar o framework PowerMock (https://github.com/powermock/powermock). Um exemplo de uso pode ser encontrado em [Mokkapaty02].

Referências

- [Mokkapaty01] Ram Mokkapaty, “Mockito Spy Example” Em: https://examples.javacodegeeks.com/core-java/mockito/mockito-spy-example/
- [Mokkapaty02] Ram Mokkapaty, “Mockito mock static method example” Em: https://examples.javacodegeeks.com/core-java/mockito/mockito-mock-static-method-example/