**Especialização em Tecnologias Aplicadas a Sistemas de Informação com Metodologias Ágeis**

**Disciplina: Design Patterns, Smells e Refactoring**

**Professor: Guilherme Lacerda**

**Workshop Ferramenta Mockito**

1. **AUTORES**

Alécio Dalprá - [aleciodalpra@gmail.com](mailto:aleciodalpra@gmail.com)

Fábio Castilhos - [fabcastilhos@gmail.com](mailto:fabcastilhos@gmail.com)

1. **INTRODUÇÃO**

O Mockito é um framework de código aberto, desenvolvido pela Google para uso em testes de unidade na linguagem Java. Possui algumas iniciativas para utilização em outras linguagens como *Python*, *Flex*, *JavaScript*, *Scala*, *Objective C*, *Pearl*, *PHP* e *MATLAB*.

Com o Mockito é possível entregar um código de teste mais fácil de ler e modificar, reduzindo ou minimizando acoplamentos. Ele simula a utilização um objeto real para uma classe específica, auxiliando nos testes de unidade quando precisamos verificar a comunicação entre objetos.

O Mockito difere um pouco dos outros frameworks, pois tenta eliminar o padrão “esperar/executar/verificar” que é seguido por grande parte dos *frameworks*. Sua utilização é bastante simplificada, e para começar a utilizá-lo basta baixar o pacote no site do projeto. Possui também uma documentação bastante útil, recheada de exemplos.

Como principais características do Mockito podemos citar:

* O setup para a criação de um *mock*, diferentemente de outros *frameworks*, é bastante simples.
* Possibilidade de criar *mocks* tanto de classes concretas como de *interfaces*.
* Criação de *mocks* por meio de anotações.
* Facilidade na verificação de erros: a verificação de testes que falham é simples, bastando olhar o *stack trace* disponibilizado.
* Permite verificar a quantidade de vezes que um método foi chamado.
* Permite a verificação por meio de matchers de argumentos.

1. **MÉTODOS E FUNÇÕES**

O Mockito fornece uma série de métodos para comportamentos de *mocks* específicos:

|  |  |
| --- | --- |
| **Método** | **Descrição** |
| thenReturn(T valueToBeReturned) | Retorna valor dado |
| thenThrow(Throwable toBeThrown)  thenThrow(Class<?extends Throwable>  toBeThrown) | Lança uma exceção |
| then(Answer answer)  thenAnswer(Answer answer) | Usa o código criado pelo usuário para responda |
| thenCallRealMethod() | Chama o método real quando trabalhando com parcial mock /spy |
| doThrow(Throwable toBeThrown)  doThrow(Class<?extends Throwable> toBeThrown) | Lança uma exceção dada |
| doAnswer(Answer answer) | Usa o código criado pelo usuário para responder |
| doCallRealMethod() | Trabalhando com espião |
| doNothing() | Faz nada |
| doReturn(Object toBeReturned) | Retorna valor dado (não para vazio métodos) |

Por padrão, o Mockito verifica se um determinado método foi chamado uma vez e apenas uma vez. Com ele é possível criar *mocks* de verificação personalizado:

|  |  |
| --- | --- |
| **Nome** | **O método de verificação foi:** |
| times(int wantedNumberOfInvocations) | Chamado exatamente ‘n’ vezes (um por padrão) fl |
| never() | Nunca foi chamado. |
| atLeastOnce() | Chamado pelo menos uma vez. |
| atLeast(int minNumberOfInvocations) | Chamado pelo menos ‘n’ vezes. |
| atMost(int maxNumberOfInvocations) | Chamado no máximo ‘n’ vezes. |
| only() | O único método chamado pelo *mock* |
| timeout(int millis) | Interagiu em um intervalo de tempo especificado. |

O Mockito também permite redefinir um valor padrão retornado nos métodos:

|  |  |
| --- | --- |
| **Resposta Padrão** | **Descrição** |
| RETURNS\_DEFAULTS | Retorna um valor "vazio" padrão (por exemplo, nulo, 0, falso, coleção vazia): usado por padrão |
| RETURNS\_SMART\_NULLS | Cria um espião de um determinado objeto |
| RETURNS\_MOCKS | Retorna um valor "vazio" padrão, mas um mock em vez de nulo |
| RETURNS\_DEEP\_STUBS | Permite um simples esboço profundo (por exemplo, Dado (ourMock.getObject (). GetValue ()). WillReturn (s)) |
| CALLS\_REAL\_METHODS | Ligue para um método real de objeto mockado |

1. **ANOTAÇÕES**

Mockito oferece três anotações - @Mock, @Spy, @Captor - para simplificar a  
processo de criação de objetos relevantes usando métodos estáticos. A anotação @InjectMocks simplifica a falsificação e a injeção de espião. Pode injetar objetos usando injeção de construtor, injeção de *setter* ou injeção de campo.

|  |  |
| --- | --- |
| **Notação** | **Responsabilidade** |
| @Mock | Cria uma *mock* de um determinado tipo. |
| Spy | Cria *spy* de um determinado objeto. |
| @Captor | Cria um argumento *captor* de um determinado tipo. |
| @InjectMocks | Cria um objeto InjectMocks de um determinado tipo. |

1. **LIMITAÇÕES**

O Mockito possui algumas limitações que merecem ser lembradas. Elas são geralmente restrições técnicas, que podem levar a criação de códigos pouco verificáveis. Com o Mockito não se deve criar:

* *Mocks* de classes finais.
* *Mocks* de *enums*.
* *Mocks* de métodos finais.
* *Mocks* de métodos estáticos.
* *Mocks* de métodos privados.
* *Mocks* de hashCode() e equals().

No entanto, ao trabalhar com um código legado mal escrito, lembre-se de que algumas das limitações mencionadas acima podem ser mitigadas usando as bibliotecas do *PowerMock* ou *JMockit*.

1. **INSTRUÇÕES DE INSTALAÇÃO**

Os artefatos Mockito estão disponíveis no *Maven Central Repository* (MCR).  
A maneira mais fácil de disponibilizar o MCR no seu projeto é colocar a seguinte configuração no seu gerenciador de dependências:

|  |  |
| --- | --- |
| **MAVEN** | <dependency>  <groupId>org.mockito</groupId>  <artifactId>mockito-core</artifactId>  <version>1.9.5</version>  <scope>test</scope>  </dependency> |
| **GRADLE** | testCompile “org.mockito: mockito −core: 1.9.5“ |
| **IVY** | <dependency org=”org.mockito” name=”mockito-core” rev=”1.9.5” conf=”test->default”/> |

Também é possível instalar o *framework* Mockito executando a sequência de passos descritos a seguir:

**Passo 1: Instalar Mockito *framework***

1. Baixar a versão mais recente do *framework* Mockito, acessando: <https://code.google.com/p/mockito/downloads/list>

2. Descompactar o pacote mockito-2.0.2.zip e salvar o arquivo .jar em sua unidade C: C: \> Mockito.

|  |  |
| --- | --- |
| **SO** | **Nome do Arquivo** |
| Windows | mockito-all-1.9.5-beta.jar |
| Linux | mockito-all-1.9.5-beta.jar |
| Mac | mockito-all-1.9.5-beta.jar |

3. Definir a variável de ambiente **Mockito\_HOME** para apontar para a localização do diretório base onde o Mockito *framework*, onde os arquivos de dependência foram armazenados:

|  |  |
| --- | --- |
| **SO** | **Saída** |
| Windows | Defina a variável de ambiente Mockito\_HOME para C: \ Mockito |
| Linux | Exportar Mockito\_HOME = / usr / local / Mockito |
| Mac | Exportar Mockito\_HOME = / Library / Mockito |

**Passo 2: Definir variável CLASSPATH**

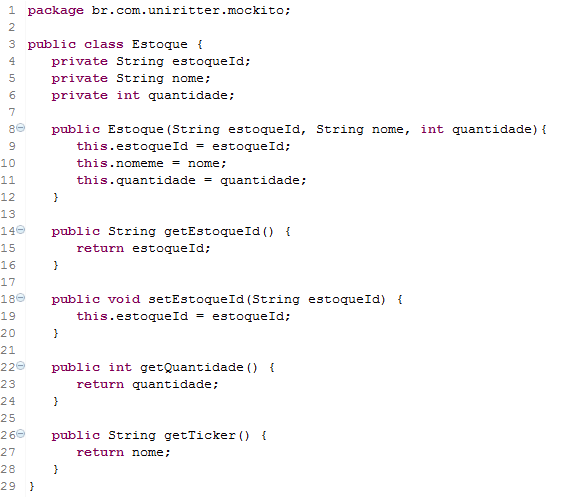
4. Definir a variável de ambiente **CLASSPATH** para apontar para o local onde o .jar do Mockito *framework* foi armazenado:

|  |  |
| --- | --- |
| **SO** | **Saída** |
| Windows | Defina a variável de ambiente CLASSPATH para% CLASSPATH%;% Mockito\_HOME% \ mockito-all-1.9.5-beta.jar;.; |
| Linux | Exportar CLASSPATH = $ CLASSPATH: $ Mockito\_HOME / mockito-all-1.9.5-beta.jar :. |
| Mac | Exportar CLASSPATH = $ CLASSPATH: $ Mockito\_HOME / mockito-all-1.9.5-beta.jar :. |

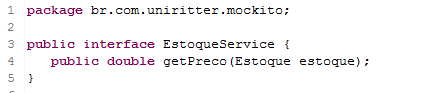
1. **EXEMPLO DE USO**
   1. **Exemplo 1: Teste de Estoque com Mockito**

O Mockito ésimples e fácil de usar. No exemplo a seguir, iremos criar uma simulação passo a passo de Estoque para obter o preço falso de alguns produtos:

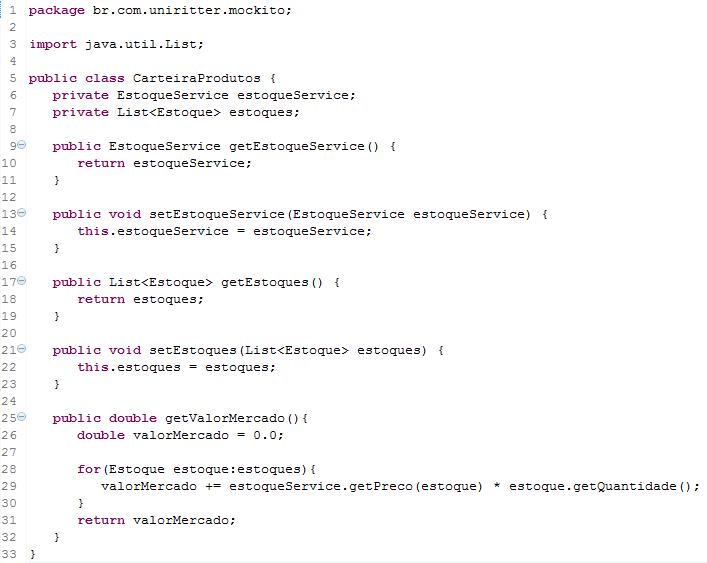
**Passo 1: Crie uma classe JAVA denominada Estoque.java**



**Passo 2: Crie uma interface denominada EstoqueService.java para obter o preço de uma ação**

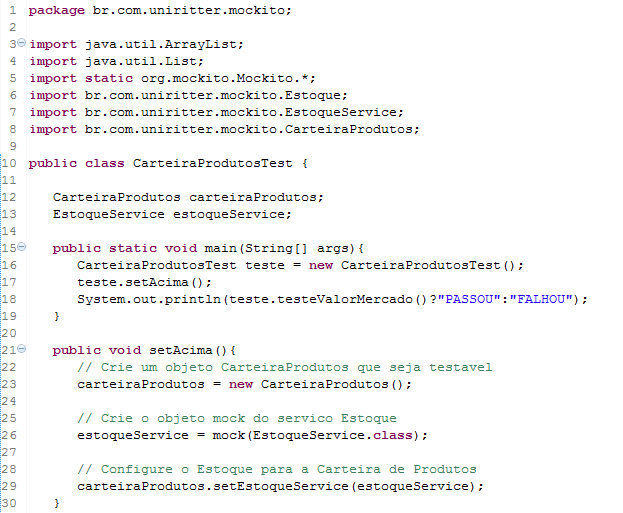


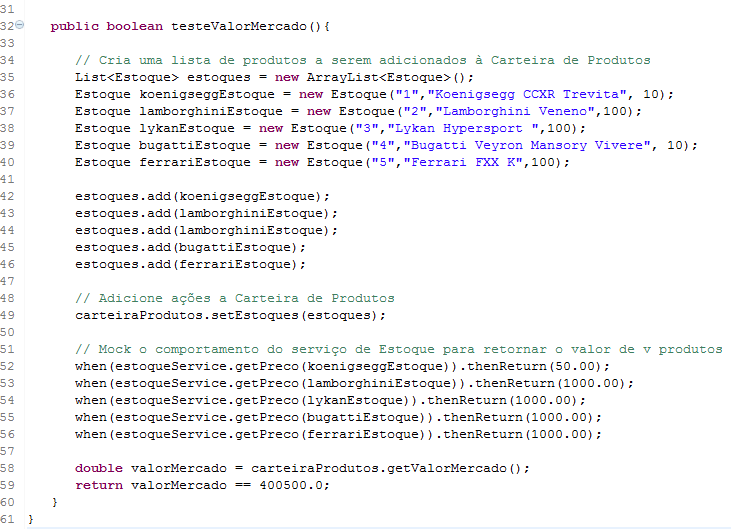
**Passo 3: Crie a classe denominada CarteiraProdutos.java para representar a carteira dos clientes**



**Passo 4: Teste a classe CarteiraProdutosTest.java**

Vamos testar a classe CarteiraProdutos, injetando nele uma falsificação de estoque. O Mockito criará simulações.





**Passo 5: Verifique o resultado**

a. Compile as classes usando o compilador javac da seguinte maneira:

**C:\Mockito\_WORKSPACE>javac Estoque.java EstoqueService.java CarteiraProdutos.java CarteiraProdutosTest.java CarteiraProdutosTest.java**

b. Agora execute o CarteiraProdutosTest. java para ver o resultado:

**C:\Mockito\_WORKSPACE>java CarteiraProdutosTest**

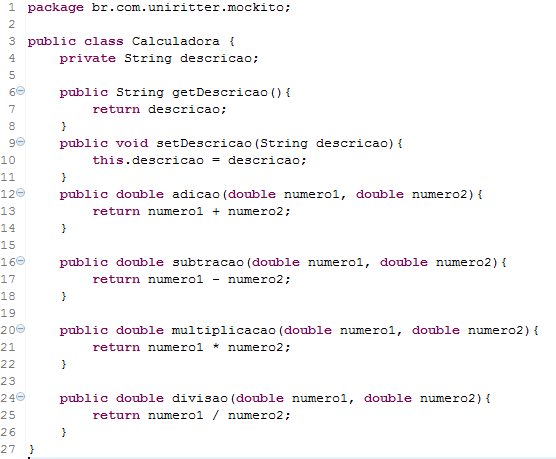
c. Verifique a saída

**PASSOU**

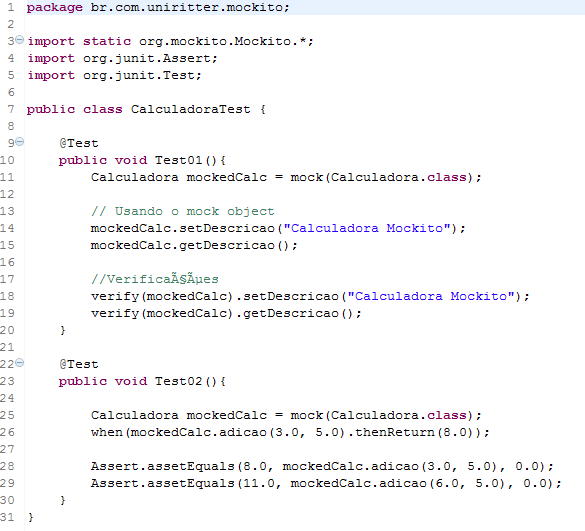
* 1. **Exemplo 2: Teste de uma Calculadora com Mockito**

Neste exemplo iremos demonstrar uma simulação de uma Calculadora utilizando o Mockito:

**Passo 1: Crie uma classe JAVA denominada Calculadora.java**



**Passo 2: Crie uma classe de teste denominada CalculadoraTests.java**



Foi adicionado comando para criar um retorno “mockado” na linha 14 e 14. Oretorno desse mock é testado nas linhas 18 e 19, onde a verificação espera que se retorne, na primeira linha “Calculadora Mockito” e na segunda nenhum valor.

O segundo teste foi “mockado” o retorno da operação de adição ao somar 3 + 5 (linha 27). Na linha 28 é feita a validação. Caso se passe valores diferentes de 3 + 5, o teste falhará, pois, o retorno esperado na linha 26 é 8.

**REFERÊNCIAS**

*Mockito framework. Tasty mocking framework for unit tests in Java*, disponível em <http://site.mockito.org/>

*DZONE. Getting Started with Mocking in Java using Mockito*, disponível em <https://dzone.com/articles/getting-started-mocking-java>

*CODE PROJECT: Mockito - a great mock framework for Java development*, disponível em <https://www.codeproject.com/Articles/516360/Mockito-a-great-mock-framework-for-Java-developmen>

*Official Mockito. Documentation Class*, disponível em <https://static.javadoc.io/org.mockito/mockito-core/2.12.0/org/mockito/Mockito.html>

*SOLID SOFT. Beyond the Mockito Refcard*, disponível em <http://blog.solidsoft.info/beyond-the-mockito-refcard/>