

Testes com Java

Trabalhando com Mocks

Rodrigo Tassini
Software Engineer - CMA





Objetivos da Aula

1. O que são Mocks?

2. Frameworks

- Mockito
- EasyMock
- PowerMockito

3. Exemplo com Mockito



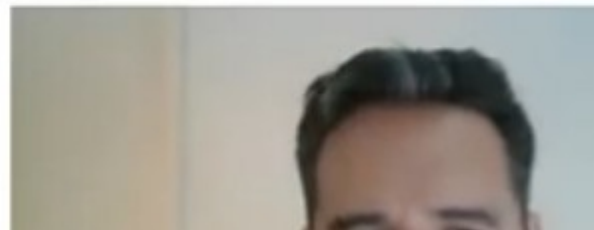
Requisitos Básicos

- ✓ Projeto da aula anterior
- ✓ IDE Eclipse
- ✓ JDK 6 ou maior
- ✓ Maven
- ✓ Junit 4
- ✓ Mockito 3



O que são Mocks?

- Objetos Mock, objetos simulados ou simplesmente Mock
- Razões para usar
- Limitações



Objeto mock

[ocultar]

Origem: Wikipédia, a enciclopédia livre.



Este artigo ou secção **não cita fontes confiáveis e independentes**. Ajude a [inserir referências](#).

O conteúdo não [verificável](#) pode ser removido.—*Encontre fontes: [Google](#) (notícias, livros e acadêmico)*
(*Agosto de 2020*)

Objetos mock, **objetos simulados** ou simplesmente **mock** (do inglês **mock object**) em [desenvolvimento de software](#) são [objetos](#) que simulam o comportamento de objetos reais de forma controlada. São normalmente criados para testar o comportamento de outros objetos. Em outras palavras, os objetos mock são objetos “falsos” que simulam o comportamento de uma classe ou objeto “real” para que possamos focar o teste na unidade a ser testada.

Razões para usar

Em **teste unitário**, pode simular o comportamento de objetos reais complexos e são, portanto, muito úteis quando os objetos reais são difíceis de criar ou impossíveis de serem incorporados no teste de unidade. Por exemplo, em um teste para um objeto que faz uma chamada remota é necessário que do outro lado exista uma previsão para a chamada. Neste caso, em vez de usar o objeto real que faz a chamada, usa-se um mock object que simula o comportamento do objeto real. Se um objeto tem alguma das características a seguir, é provável que mock objects possam ser utilizados em seu lugar:

- gera resultados não determinísticos (e.g. a hora ou temperatura atual);
- tem estados que são difíceis de criar ou reproduzir (e.g. erro de comunicação da rede);
- é lento (e.g. um banco de dados completo que precisa ser inicializado antes do teste);
- ainda não existe ou pode ter comportamento alterado;
- teriam que adicionar informações e métodos exclusivamente para os testes (e não para sua função real).

Por exemplo, um programa de despertador que faz uma campainha tocar em um certo momento necessita obter a hora atual. Para testar este comportamento, o teste de unidade deve esperar até que a hora programada chegue, para testar a campainha de forma correta. Se um mock object for usado no lugar do objeto real, então ele pode ser programado para simular a hora de programação do despertador. Assim o código do despertador pode ser testado corretamente.

Um objeto mock tem precisamente o modelo do comportamento do objeto que ele está simulando ("*mocking*"). O comportamento correto pode se tornar difícil de se obter se o objeto que está sendo simulado ("*mocked*") foi feito por outro desenvolvedor, ou, quando esse não tenha sido escrito ainda. Se o comportamento não for modelado corretamente, então os testes unitários devem registrar que o teste passou mesmo se uma falha ocorresse no tempo de execução, nas mesmas condições que o teste está exercendo e dessa forma, tornando inútil o teste unitário.

Ferramentas de Auxílio para Criação de Mocks


Os `Mocks` possuem bibliotecas que ajudam na sua criação. A sintaxe varia de ferramenta para ferramenta, mas no geral todas são bastante agradáveis, simples e possuem uma boa cobertura para testes de diferentes níveis.

Diversas linguagens de programação ou ambientes de desenvolvimento possuem bibliotecas para criar Mocks.

- Para Java temos o `JMockit`, `Mockito`, `EasyMock`, `JMock`, `MockCreator`, `MockLib` e `HibernateMock`.
- Para plataforma .NET temos o `NMockLib`, `Rhino Mocks`, `NMock` e `NMock 2 TypeMock`.
- Para o Ruby temos o `Mocha`, `RSpec` e `FlexMock`.
- Para linguagem PHP temos o `SimpleTest`, `Yay! Mock`, `SnapTest` e `PHPUnit`.
- Para o Delphi, `Kylix`, `FreePascal` ou `Pascal Mock`.

Frameworks

- Mockito
- EasyMock
- PowerMockito

Mockito é um framework de teste de código aberto para Java lançado  sob a licença MIT. [3] [4] A estrutura permite a criação de objetos duplos de teste (objetos simulados) em testes de unidade automatizados com a finalidade de desenvolvimento orientado a teste (TDD) ou desenvolvimento orientado a comportamento (BDD).

O nome e o logotipo do framework são uma brincadeira com mojitos, um tipo de bebida.

Mockito

Mockito é mais um framework de código aberto para testes. O Mockito está sob a licença MIT License. O Mockito difere um pouco dos outros frameworks porque ele tenta eliminar o padrão expect-run-verify que é seguido por grande parte dos frameworks como vimos anteriormente. Assim ele não estabelece expectativas antecipadamente, e com isso temos um acoplamento reduzido ou minimizado, um código de teste mais fácil de ler e modificar.

O link do download do Mockito pode ser realizado na área de Links. Após isso basta descompactar os arquivos e colocar o Jar mockito-all-1.9.5.jar no classpath do projeto.

Vamos utilizar o mesmo exemplo anterior, porém agora utilizando Mockito. Segue na **Listagem 10** o código de exemplo em Mockito.

Mockito

Exemplo 1

Um serviço que faz uma validação de campos obrigatórios, e se satisfatório salva o registro em um repositório, caso contrário lança uma Exception.

Abaixo foi declarado uma classe **PessoaService** que faz a regra que será testada:

```
@Service
public class PessoaService {

    @Autowired
    private PessoaRepository pessoaRepository;

    public void save (Pessoa pessoa) {
        validarCampos(pessoa);
        pessoaRepository.save(pessoa);
    }

    private void validarCampos(Pessoa pessoa) {
        if (pessoa.getEmail() == null || pessoa.getNome() == null)
            throw new RuntimeException("Campos obrigatórios...");
    }
}
```

Classe PessoaService

O Teste unitário da classe:

```
@RunWith(MockitoJUnitRunner.class)
public class PessoaServiceTest {

    @Mock
    private PessoaRepository pessoaRepository;

    @InjectMocks
    private PessoaService pessoaService;

    @Test
    public void testSalvarComSucesso () {
        Pessoa pessoa = new Pessoa();
        pessoa.setNome("Teste");
        pessoa.setEmail("teste@gmail.com");
        pessoaService.save(pessoa);
        verify(pessoaRepository, times(wantedNumberOfInvocations: 1)).save(pessoa);
    }

    @Test(expected = Exception.class)
    public void testCamposObrigatoriosInvalidos () {
        Pessoa pessoa = new Pessoa();
        pessoaService.save(pessoa);
    }
}
```

Teste unitário da classe PessoaService

Para que os testes possam rodar com o Mockito a classe de teste deve estar configurada para tal, isso é feito através da anotação @ ***RunWith*** que faz referência a classe ***MockitoJUnitRunner.class***.

```
@RunWith(MockitoJUnitRunner.class)
public class PessoaServiceTest
```

Para entender mais sobre Runners do JUnit [clique aqui](#)

Em seguida vem os objetos a serem mockados, no caso ***PessoaRepository***, através da anotação @ ***Mock***. O Mockito vai criar uma cópia da estrutura dessa classe com uma implementação vazia e com retornos padrões em todos os métodos. Veja que não é necessário iniciar o objeto. O Mock pode ser feito tanto com classes concretas, abstratas e interfaces.

```
@Mock
private PessoaRepository pessoaRepository;
```

EasyMock é uma estrutura de teste de código aberto para Java lançado sob a licença Apache. [2] O framework permite a criação de objetos duplos de teste com a finalidade de Test-driven Development (TDD) ou Behavior Driven Development (BDD). [3]



Pesquisa realizada em 2013 em 10.000 projetos GitHub descobriu que EasyMock é a 32ª biblioteca Java mais popular. [4]

O EasyMock fornece objetos Mock gerados dinamicamente (em tempo de execução), sem a necessidade de implementá-los. No EasyMock, a definição de Objeto Mock é diferente de usar um Objeto Mock implementado. Objetos de simulação são construídos em tempo de execução e implementações adicionais não podem ser definidas para esses objetos. [5]



EasyMock

O `EasyMock` é uma biblioteca que fornece uma forma fácil para usarmos Mock Objects para classes ou interfaces. O `EasyMock` está disponível pela [licença Apache 2](#). Para utilizar o `EasyMock` devemos utilizar a versão `Java 1.5.0` ou acima.

O `EasyMock` está disponível no repositório central do `Maven` apenas adicionando a dependência da **Listagem 1**.

```
1 <dependency>
2   <groupId>org.easymock</groupId>
3   <artifactId>easymock</artifactId>
4   <version>3.2</version>
5   <scope>test</scope>
6 </dependency>
```

Listagem 1. Dependência para EasyMock

JMock

JMock é uma biblioteca para Mock Objects que ajuda os desenvolvedores a projetar testes para interações entre os objetos. A biblioteca JMock permite fazermos de forma fácil e rápida a definição de Mock Objects, também permite que possamos fazer de forma mais precisa a especificação das interações entre os objetos reduzindo a fragilidade dos testes, além disso o JMock trabalha bem com autocomplemento, refatorações e é considerado fácil de estender.

O download do JMock está disponível na área de **Links**. Para instalar o JMock basta descompactar os arquivos e colocar os JARs `jmock-2.6.0.jar`, `hamcrest-core-1.3.jar` e `hamcrest-library-1.3.jar` no `classpath` do projeto.

Na **Listagem 8** será demonstrado o mesmo exemplo anterior, porém agora utilizando JMock.

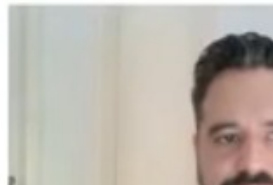


DIGITAL
INNOVATION
ONE

Testes com Java

Explorando o mundo dos Asserts

Rodrigo Tassini
Software Engineer - CMA



Enfim, as Assertions servem para testar se o seu código esta de acordo com as regras que deve seguir (vamos dar exemplos melhores a frente), sem precisar fazer laços enormes com “`throw new`” e mais outros recursos só para testar se tudo está certo. As Assertions são utilizadas SOMENTE e APENAS para o ambiente de desenvolvimento, quando o software for para produção, elas devem ser retiradas sem que o software seja afetado. Por isso ao implementar `assert` não deixe seu software “dependente” destes, de forma que depois você não consiga tirar sem mudar a lógica do mesmo.

A sintaxe para uso de Assertions pode ser de duas formas:

- `assert Expressão1 ;`
- `assert Expressão1 : Expressão2 ;`

Conclusão

Assertivas ou Asserts é um recurso muito poderoso para aumentar a produtividade no desenvolvimento de sistemas, assim você não precisará ficar debugando linhas e linhas de código para descobrir 1 simples erro. Se utilizado da forma correta, com certeza será um forte aliado em grandes projetos.

Acontece que o assert sempre deverá ter um valor Verdadeiro, caso contrário ele levantará uma exceção `AssertionError`. No primeiro caso de sintaxe é testado apenas se a expressão passada é `True`, caso contrário é “levantada” uma exceção, já no segundo caso além da expressão principal, uma segunda expressão serve como mensagem para que o programador veja o que esta ocorrendo.



ViniGodoy

Abr '07

Serve para você testar pré ou pós condições de sua função, deixando-as explícitas em seu código.

Geralmente, condições desse tipo só tem sentido no momento da programação, não no código do usuário final, por isso, as asserções só estarão disponíveis se a flag `-ea` for utilizada na VM.

Tem um tutorial do GUJ sobre isso, [aqui](#) 114.

Requisitos Básicos

- ✓ Projeto da aula anterior
- ✓ IDE Eclipse
- ✓ JDK 6 ou maior
- ✓ Maven
- ✓ Junit 4
- ✓ Hamcrest
- ✓ AssertJ

Asserts

- Definição
- Como usar:

```
Assert.assertEquals(...)
```

```
import static org.junit.Assert.*;
```

```
...  
AssertEquals(...);
```

- Métodos



Hamcrest e Matcher

- Hamcrest
 - Definição
 - Setup
 - Exemplo
 - Matcher
 - Object Matcher
 - Bean Matcher
 - Collection Matcher
 - Number Matcher
 - Text Matcher
-

Bibliotecas de Matcher

Como foi mencionado anteriormente, as asserções providas pelo JUnit ou TestNG não são flexíveis o suficiente. No mundo Java há pelo menos duas bibliotecas open source que suprem essas necessidades: [AssertJ](#) (um fork do projeto FEST Fluent Assertions) e o [Hamcrest](#). Os dois são bastante poderosos e permitem atingir efeitos similares. A principal diferença do AssertJ sobre o Hamcrest é sua API, baseada em interfaces fluentes, que é perfeitamente suportada pelas IDEs.

Para integrar o AssertJ com o JUnit ou TestNG basta somente realizar os imports necessários, parar de utilizar as asserções padrões do framework de teste e iniciar o uso das asserções fornecidas pelo AssertJ.

O AssertJ fornece muitas asserções que seguem o mesmo padrão: começam com o método `assertThat()`, um método estático da classe `Assertions` que recebe o objeto testado como parâmetro. Logo em seguida são os métodos de asserção, no qual cada um verifica as propriedades do objeto testado:

```
assertThat(myDouble).isLessThanOrEqualTo(2.0d);

assertThat(myListOfStrings).contains("a");
assertThat("some text")
    .isNotEmpty()
    .startsWith("some")
    .hasLength(9);
```


O AssertJ provê um conjunto muito mais rico de asserções que o JUnit ou TestNG. Pode-se encadear os métodos entre si, assim como é mostrado no exemplo anterior, permitindo que a IDE sugira os métodos que podem ser utilizados baseado no tipo de objeto sendo testado. Por exemplo, no caso de uma variável de ponto flutuante, após `assertEquals(myDouble)` e CTRL+SPACE pressionado (dependendo da IDE), será exibido uma lista de métodos como `isEqualTo(expectedDouble)`, `isNegative()` ou `isGreaterThan(otherDouble)` - todos que fazem sentido com a verificação do valor de ponto flutuante.

```
import static org.assertj.core.api.Assertions.assertThat;
```

```
public class SampleTest {
```

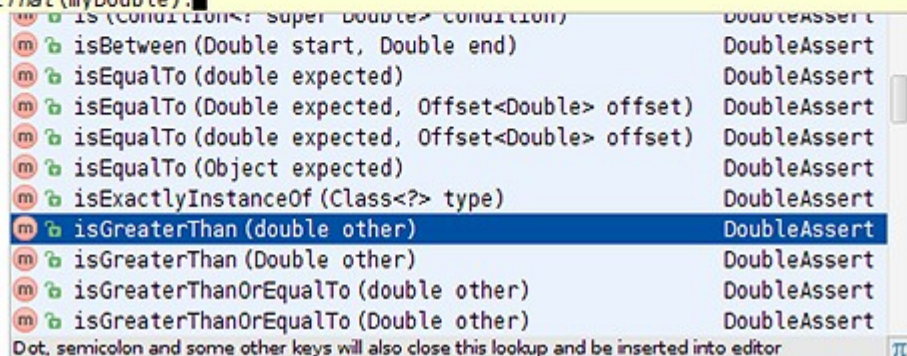
```
    @Test
```

```
    public void sampleTestMethod() {
```

```
        double myDouble = 1.2;
```

```
        assertThat(myDouble).█
```

```
    }  
}
```



Asserts são fornecidos por qual classe?

com.java.Assert

org.junit.Assert ✓

Nenhuma das alternativas.

java.util.Assert

com.junit.Assert

PRÓXIMA PERGUNTA

Qual das afirmações abaixo sobre limitações no uso de Mocks é verdadeira?

Tipos primitivos, Classes e Collections.

Classes, Métodos e Objetos.



Variáveis, Instâncias e Objetos.

Tipos primitivos, Objetos e Arrays.



Listas e Collections.

O Matcher `greaterThan` verifica se:

Nenhuma das alternativas.

Verifica se o valor esperado é maior que o atual.

Verifica se o valor esperado é maior ou igual ao atual.



Verifica se o valor atual é maior ou igual ao esperado.

Verifica se o valor atual é maior que o esperado.



O que é AssertJ?

Um framework usado para escrever Assertions para alguns objetos.



Um framework usado para escrever Assertions em linguagem estruturada.

Nenhuma das alternativas.

Um framework usado para escrever Assertions em português.

Um framework usado para escrever Assertions mais fluentes.



PRÓXIMA PERGUNTA

O que são mocks?

Nenhuma das alternativas.

Objetos instanciados pela JVM.

Objetos alocados dentro do pacote de testes.

Objetos que simulam o comportamento de objetos reais de forma controlada.



Objetos que controlam o comportamento de objetos reais.

PRÓXIMA PERGUNTA

O que são mocks?

Nenhuma das alternativas.

Objetos instanciados pela JVM.

Objetos alocados dentro do pacote de testes.

Objetos que simulam o comportamento de objetos reais de forma controlada.



Objetos que controlam o comportamento de objetos reais.

PRÓXIMA PERGUNTA

O framework Mockito permite que:

Sejam construídos testes unitários.



Sejam construídos testes para APIs.

Sejam construídos testes com uma API simples e clara.



Nenhuma das alternativas.

Sejam construídos testes com classes Java nativas.

PRÓXIMA PERGUNTA

Qual das afirmações abaixo sobre razões para se usar Mocks é verdadeira?

Uma das razões para usar Mocks é que criar objetos reais para testes é difícil.



Uma das razões para usar Mocks é que são objetos flexíveis.

Uma das razões para usar Mocks é que podem ser criados e destruídos em tempo de execução.



Uma das razões para usar Mocks é que não são necessários frameworks adicionais.

Nenhuma das alternativas.

PRÓXIMA PERGUNTA

AssertEquals verifica se:

O valor atual e esperado não são iguais.

Nenhuma das alternativas.

O valor atual e esperado são iguais.

O valor esperado e atual não são iguais.

O valor esperado e atual são iguais.



Qual Matcher permite inspecionar classes da API de Collection?

Object Matcher

Nenhuma das alternativas.

Collection Matcher



Number Matcher

Bean Matcher

FINALIZAR