

# Mockito

VALMIR JUNIOR



1. Conceito de  
mockito

2. Tipos de mocks  
e como utilizar

3. Exemplos no  
Liferay

# conceito sobre mockito



O Mockito é um framework de testes unitários e o seu principal objetivo é instanciar classes e controlar o comportamento dos métodos.

# Quando devo utilizar mocks

---

- Quando queremos simular o comportamento de um objeto real durante os testes, de forma a isolá-lo e testá-lo de suas dependências externas.

# Analogia com mundo real

Caso 1



Caso 2



# Formas de criar mocks

Exemplo.java

```
1 Mockito.mock(EmployeeReposit.class);  
2  
3 @Mock  
4 private EmployeeRepository employeeRepository;
```

# Configurando a classe testada

```
Exemplo.java  
1 @InjectMocks  
2 private EmployeeController employeeController;
```

O Mockito vai criar uma instância real dessa classe e injetar todos os objetos @ Mock que foram declarados na classe de teste.

# Habilitando as anotações

Para essas anotações `@Mock` e `@InjectMocks` funcionar, é preciso habilitá-las. existem duas formas:



Exemplo.java

```
1 @RunWith(MockitoJUnitRunner.class)
2 public class EmployeeControllerTest {
3     //Anotando a classe de teste com @RunWith(MockitoJUnitRunner.class)
4
5 }
6
7 //Usando o MockitoAnnotations.initMocks () antes dos testes
8 @Before
9 public void setup() {
10     MockitoAnnotations.initMocks(this);
11 }
```

# Verify

```
1 // verificar se determinado método de um mock foi executado:  
2  
3 //leilao é uma classe que foi mockada.  
4 Mockito.verify(leilao).salvar();  
5  
6 //É possível fazer verificar por quantidade de interação:  
7 Mockito.verify(synonymSetStorageAdapter, Mockito.times(2)  
8     .delete(  
9         Mockito.any(), Mockito.anyString()  
10    );  
11  
12 //Verificar quando não houver interação:  
13 Mockito.verifyNoInteractions(_journalFolderLocalService);
```

# When

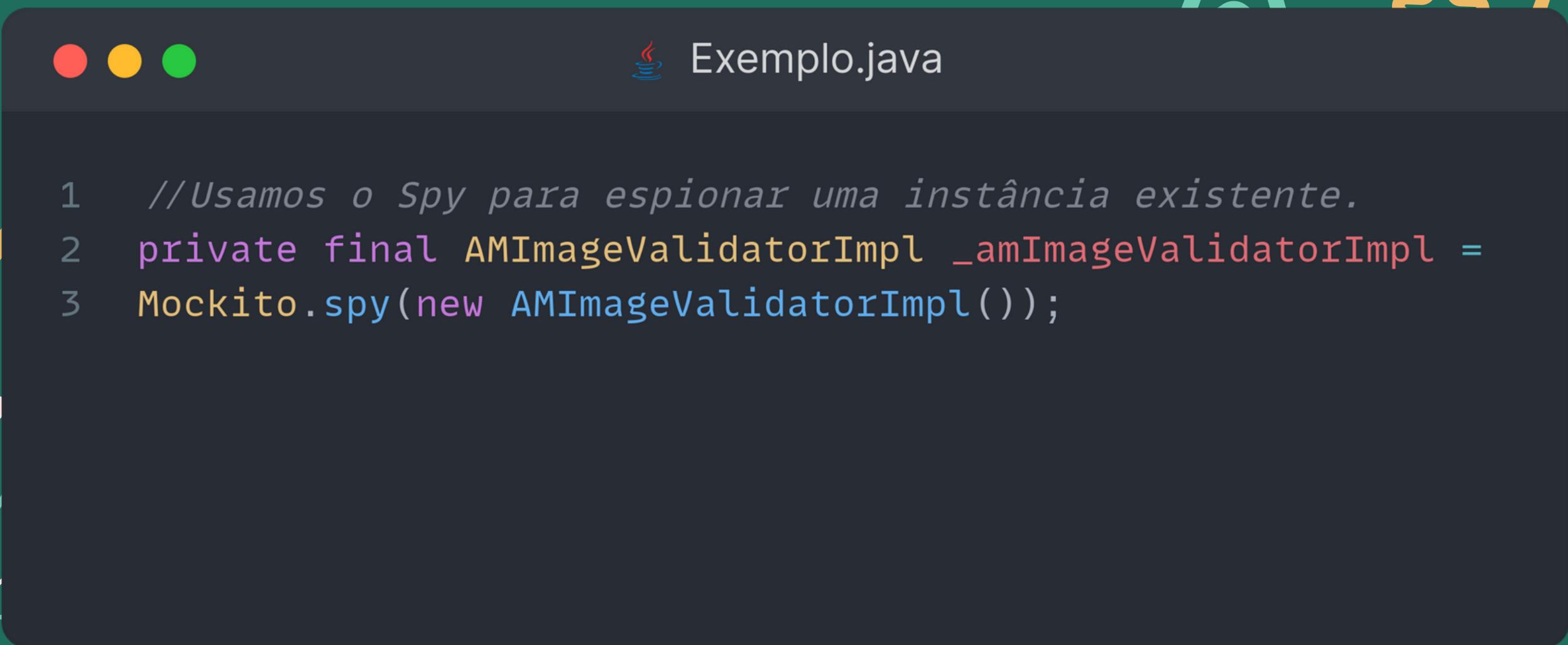
```
1 // Serve para manipular o nosso mock, isto é, para dizer:  
2 // "Mockito, quando (when) esse método do mock for chamado, devolva isso  
3  
4 Mockito.when(_portletPreferences.getValue("enableComments", null)  
5     ).thenReturn(  
6         "false"  
7     );  
8
```

# Verificando comportamento em caso de exceções

```
Exemplo.java
```

```
1  private void _whenGetEntryThenThrow(Throwable throwable) throws Exception {
2      Mockito.when(
3          _actionRequest.getParameter(Mockito.anyString())
4      ).thenReturn(
5          String.valueOf(10L)
6      );
7
8      Mockito.when(
9          _blogsEntryService.getEntry(Mockito.anyLong())
10     ).thenThrow(
11         throwable
12     );
13 }
```

# SPY



Exemplo.java

```
1 //Usamos o Spy para espionar uma instância existente.  
2 private final AMImageValidatorImpl _amImageValidatorImpl =  
3 Mockito.spy(new AMImageValidatorImpl());
```

# Exemplo de test



```
1 @Test
2 public void spyTest() {
3     //Employee = Empregado
4     List<Employee> employees = Mockito.spy(new ArrayList<Employee>());
5     Employee one = new Employee("Bruce Wayne", "CEO");
6     Employee two = new Employee("Clarck Kent", "CTO");
7
8     employees.add(one);
9     employees.add(two);
10
11    Mockito.verify(employees).add(one);
12    Mockito.verify(employees).add(two);
13    assertEquals(2, employees.size());
```

# captor

```
Exemplo.java

1 import org.junit.Test;
2 import org.junit.runner.RunWith;
3 import org.mockito.Captor;
4 import org.mockito.Mock;
5 import org.mockito.junit.MockitoJUnitRunner;
6
7 import java.util.List;
8
9 import static org.mockito.Mockito.verify;
10
11 @RunWith(MockitoJUnitRunner.class)
12 public class ExampleTest {
13
14     @Mock
15     private List<String> mockList;
16
17     @Captor
18     private ArgumentCaptor<String> stringCaptor;
19
20     @Test
21     public void testMethodCall() {
22         // Chama algum método no mockList com um argumento
23         mockList.add("Teste");
24
25         // Usa o captor para capturar o argumento passado no método add
26         verify(mockList).add(stringCaptor.capture());
27
28         // Realiza verificações sobre o argumento capturado
29         String capturedArgument = stringCaptor.getValue();
30         // Faça algo com o argumento, por exemplo, assert ou outras verificações
31     }
32 }
33 }
```

É usado para capturar um determinado objeto. Isso é útil quando você precisa verificar ou realizar verificações mais detalhadas sobre os argumentos passados a um método.

# ArgumentMatchers

Exemplo.java

```
1  /* São expressões que você pode usar como
2   * argumentos em verificações ou comportamentos de objetos mockados */
3
4  // Verifica se o método cadastrar foi chamado com qualquer argumento de String
5  Mockito.verify(mockObj).cadastrar(any(String.class));
6
7  // Verifica se o método convidar foi chamado com a string "example"
8  Mockito.verify(mockObj).convidar(eq("example"));
9
10 // Verifica se o método depositar foi chamado com qualquer argumento do tipo int
11 Mockito.verify(mockObj).depositar(anyInt());
12
13 // Verifica se o método sacar foi chamado com um argumento nulo
14 Mockito.verify(mockObj).sacar(isNull());
15
16 // Verifica se o método senha foi chamado com uma string que tem pelo menos 5 caracteres
17 Mockito.verify(mockObj).senha(matches(s → s.length() ≥ 5));
```

# Referências

---

- <https://dev.to/daienelima/mockito-como-utilizar-de-maneira-simples-4h86>
- <https://www.alura.com.br/>
- <https://medium.com/cwi-software/testando-seu-c%C3%B3digo-java-com-o-mockito-framework-8bea7287460a>
- <https://site.mockito.org/>
- <https://inside.contabilizei.com.br/conceitos-basicos-sobre-mockito-73b931ce0c2c>

# Fim!!!

Boas festas!

