

CURSO DE TESTES AUTOMATIZADOS

TÓPICOS AVANÇADOS

Wagner Costa

wcaquino@gmail.com

OBJECT MOTHERS & TEST DATA BUILDER PATTERNS

O PROBLEMA

```
@Test
public void TestVerificarEntidadeValida() throws Exception{
    Entidade entidade = new Entidade();
    entidade.setNome("Nome Entidade");
    entidade.setNumeroDocumento(123L);
    entidade.setTipoDocumento(2);
    entidade.setDataInicial(new Date());
    entidade.setDataFinal(new Date());
    entidade.setEmail("asd@asd.com");
    assertTrue(exercicioJUnit.verificarEntidadeValida(entidade));
}
```

- Zero Reúso
- Manutenção cara
- Complexidade alta

EXTRAIR PARA REUSAR...

```
@Test
public void TestVerificarEntidadeValida() throws Exception{
    Entidade entidade = getEntidadeValida();
    assertTrue(exercicioJUnit.verificarEntidadeValida(entidade));
}

private Entidade getEntidadeValida() {
    Entidade entidade = new Entidade();
    entidade.setNome("Nome Entidade");
    entidade.setNumeroDocumento(123L);
    entidade.setTipoDocumento(2);
    entidade.setDataInicial(new Date());
    entidade.setDataFinal(new Date());
    entidade.setEmail("asd@asd.com");
    return entidade;
}
```

- Reúso a nível de classe
- Manutenção reduzida
- Complexidade baixa

OBJECT MOTHERS

```
@Test
public void TestVerificarEntidadeValida() throws Exception{
    Entidade entidade = EntidadeMother.createEntidade();
    assertTrue(exercicioJUnit.verificarEntidadeValida(entidade));
}
```

```
@Test
public void TestVerificarEntidadeValidaNomeVazio() throws Exception{
    Entidade entidade = EntidadeMother.createEntidade();
    entidade.setNome(null);
    assertTrue(exercicioJUnit.verificarEntidadeValida(entidade));
}
```

- Alto reúso
- Complexidade baixa
- Manutenção centralizada

TEST DATA BUILDER

```
@Test
public void TestVerificarEntidadeValida() throws Exception{
    Entidade entidade = EntidadeBuilder.umaEntidade().agora();
    assertTrue(exercicioJUnit.verificarEntidadeValida(entidade));
}
```

```
@Test
public void TestVerificarEntidadeValidaNomeVazio() throws Exception{
    Entidade entidade = umaEntidade().comNomeVazio().agora();
    assertTrue(exercicioJUnit.verificarEntidadeValida(entidade));
}
```

- Alto reúso
- Complexidade baixa
- Manutenção centralizada
- Fluência na leitura

FLUENT INTERFACE

- Implementação de uma API orientada a objetos que tem como foco prover código mais legível.
- Apresentada por Eric Evans e Martin Fowler
- Normalmente implementado usando *method chaining*.
- Principais características
 - Definido através do valor de retorno da chamada do método
 - Auto referência, onde o novo contexto é equivalente ao último
 - Terminado através de um método void.

CLASSES FAMOSAS QUE USAM FLUENT

■ JPA - EntityManager

- `em.createNamedQuery("Usuario.findByName")`
 - `.setParameter("nome", "João")`
 - `.setParameter("sobrenome", "Lenon")`
 - `.setFirstResult(1)`
 - `.setMaxResults(30)`
 - `.getResultList();`

■ EasyMock

- `EasyMock.expect(persistencia.salvar(entidade))`
 - `.andReturn(entidadePersistida). atLeastOnce();`

■ Mockito

- `when(persistencia.salvar(entidade)).thenReturn(entidadePersistida);`

CLASSE COM FLUENT

Instância privada do objeto desejado

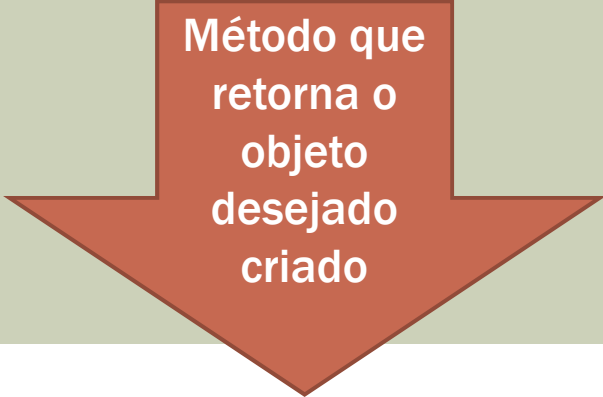
Criação do objeto através de método estático

Métodos de parametrização retornando a instância da própria classe

```
public class SubContaFI {  
  
    private SubConta subConta;  
  
    private SubContaFI() {}  
  
    public static SubContaFI UmaSubConta() {  
        SubContaFI fluente = new SubContaFI();  
        fluente.subConta = new SubConta();  
        return fluente;  
    }  
  
    public SubContaFI comId(Long id) {  
        subConta.setId(id);  
        return this;  
    }  
  
    public SubContaFI comNome(String nome) {  
        subConta.setNome(nome);  
        return this;  
    }  
  
    public SubContaFI comSaldoInicial(Double valor) {  
        subConta.setSaldoInicial(valor);  
        return this;  
    }  
  
    public SubContaFI daConta(Conta conta) {  
        subConta.setConta(conta);  
        return this;  
    }  
}
```

Construtor Privado

RECEBENDO OBJETO MONTADO



Método que
retorna o
objeto
desejado
criado

```
public SubConta agora() {  
    return subConta;  
}
```

OBJETOS ESPECÍFICOS

```
SubConta subcontaQualquer = UmaSubConta().qualquer().agora();

SubConta subcontaQualquerNaoPersistida =
    UmaSubConta().qualquer().comId(null).agora();

SubConta subcontaEspecificas =
    UmaSubConta().comId(8L).comNome("SubConta Específica").comSaldoInicial(1000D).agora();
```

Sem o fluente, seria assim...

```
SubConta subcontaEspecificas = new SubConta();
subcontaEspecificas.setId(8L);
subcontaEspecificas.setNome("SubConta Específica");
subcontaEspecificas.setSaldoInicial(1000D);
```

ESPECIFICANDO CENÁRIOS

- UmaSubConta().qualquer().agora();

```
public SubContaFI qualquer() {  
    comId(1L);  
    comNome("Conta Teste");  
    comSaldoInicial(500D);  
    daConta(UmaConta().qualquer().agora());  
    return this;  
}
```

- UmaSubConta().qualquer().comSaldoNegativo().agora();

```
public SubContaFI comSaldoNegativo() {  
    comSaldoInicial(-1D);  
    return this;  
}
```

The image features the Mockito logo, which consists of a large orange rectangle on the left and a smaller dark gray rectangle on the right. The word "MOCKITO" is written in white, bold, uppercase letters on the orange background.

MOCKITO

O QUE É MOCKITO?

- Framework para criação de mocks.
- Criado por Szczepan Faber em 2008.
- Idealizado para ser a “evolução” do EasyMock.



COMO INSTALAR

- Obter o jar do mockito em <http://code.google.com/p/mockito/>
 - Adicionar o jar mockito-all-<versão>.jar no classpath do projeto

COMO CRIAR UM MOCK?

```
@Before
public void methodSetUp() {
    negocio = new EntidadeNegocio();
    persistencia = Mockito.mock(EntidadeDAOInterface.class);
    negocio.setPersistencia(persistencia);
}
```


GRAVANDO EXPECTATIVAS

- When(mock.someAction).then...

```
@Test
public void testGetById() throws Exception{
    Entidade entidade = getEntidadeValida();

    Mockito.when(persistencia.getById(1L)).thenReturn(entidade);

    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1: " + entidade);
    entidade = negocio.getById(2L);
    System.out.println( "Entidade 2: " + entidade);
}
```

THEN WHAT?

- `then(Answer<?> answer) : OngoingStubbing<Entidade> - OngoingStubbing`
- `thenReturn(Answer<?> answer) : OngoingStubbing<Entidade> - OngoingStubbing`
- `thenCallRealMethod() : OngoingStubbing<Entidade> - OngoingStubbing`
- `thenReturn(Entidade value) : OngoingStubbing<Entidade> - OngoingStubbing`
- `thenReturn(Entidade value, Entidade... values) : OngoingStubbing<Entidade> - OngoingStubbing`
- `thenThrow(Class<? extends Throwable> ... throwableClasses) : OngoingStubbing<Entidade>`
- `thenThrow(Throwable... throwables) : OngoingStubbing<Entidade> - OngoingStubbing`

QUESTÃO

- Esse teste funciona?
 - O que será impresso?

```
@Test
public void testGetById() throws Exception{
    Entidade entidade = getEntidadeValida();

    Mockito.when(persistencia.getById(1L)).thenReturn(entidade);

    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1: " + entidade);
    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1 novamente: " + entidade);
    entidade = negocio.getById(2L);
    System.out.println( "Entidade 2: " + entidade);
}
```

RESPOSTA

```
Entidade 1: br.treinamento.Entidade@a62606f5  
Entidade 1 novamente: br.treinamento.Entidade@a62606f5  
Entidade 2: null  
|
```

- Não existe limite para utilização de cada expectativa
- Caso seja realizado uma chamada sem expectativa, o mock retornará null (ou o retorno apropriado para tipos primitivos)
- Expectativas podem ser sobrescritas, a que será utilizada será a última expectativa definida

VERIFY

- Permite checar se determinada chamada foi realizada

```
@Test
public void testGetById() throws Exception{
    Entidade entidade = getEntidadeValida();

    Mockito.when(persistencia.getById(1L)).thenReturn(entidade);

    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1: " + entidade);
    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1 novamente: " + entidade);
    entidade = negocio.getById(2L);
    System.out.println( "Entidade 2: " + entidade);

    Mockito.verify(persistencia).getById(2L);
}
```

QUESTÃO

■ Esse método funciona?

```
@Test
public void testGetById() throws Exception{
    Entidade entidade = getEntidadeValida();

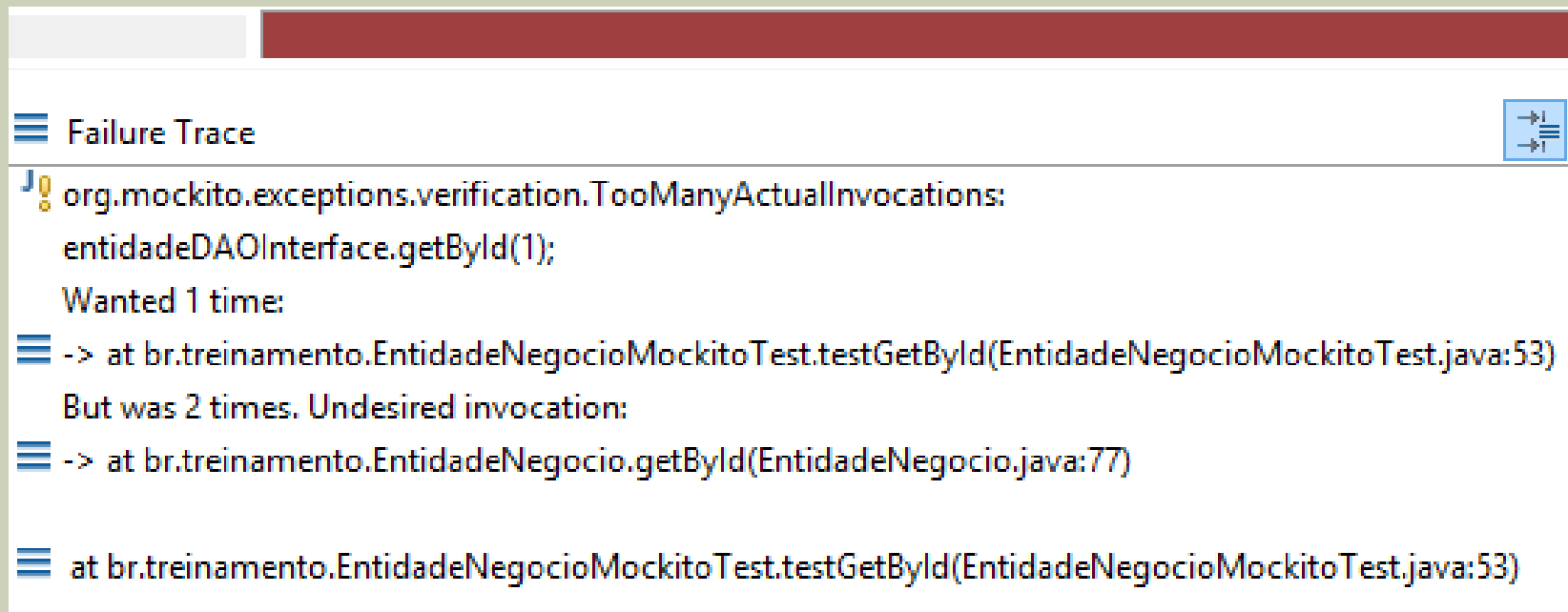
    Mockito.when(persistencia.getById(1L)).thenReturn(entidade);

    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1: " + entidade);
    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1 novamente: " + entidade);
    entidade = negocio.getById(2L);
    System.out.println( "Entidade 2: " + entidade);

    Mockito.verify(persistencia).getById(1L);
}
```

RESPOSTA

- Apesar de não precisar informar a quantidade de vezes que pode ser usado nas expectativas, o verify se importa com a quantidade de vezes que o método foi chamado.



The screenshot shows a failure trace in an IDE. At the top, there is a red progress bar. Below it, a tab labeled "Failure Trace" is visible. The main area displays the following text:

```
org.mockito.exceptions.verificatio...TooManyActualInvocations:
  entidadeDAOInterface.getByld(1);
  Wanted 1 time:
  -> at br.treinamento.EntidadeNegocioMockitoTest.testGetByld(EntidadeNegocioMockitoTest.java:53)
  But was 2 times. Undesired invocation:
  -> at br.treinamento.EntidadeNegocio.getByld(EntidadeNegocio.java:77)

  at br.treinamento.EntidadeNegocioMockitoTest.testGetByld(EntidadeNegocioMockitoTest.java:53)
```

CONFIGURANDO QUANTIDADES

- `atLeastOnce()` – pelo menos uma
- `atLeast(x)` – pelo menos “X”
- `atMost(x)` – no máximo “X”
- `Times(x)` – “X” vezes
- `Never()` – Nunca será chamado

```
@Test
public void testGetById() throws Exception{
    Entidade entidade = getEntidadeValida();

    Mockito.when(persistencia.getById(1L)).thenReturn(entidade);

    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1: " + entidade);
    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1 novamente: " + entidade);
    entidade = negocio.getById(2L);
    System.out.println( "Entidade 2: " + entidade);

    Mockito.verify(persistencia, times(2)).getById(1L);
    Mockito.verify(persistencia).getById(2L);
    Mockito.verify(persistencia, never()).getById(3L);
}
```


CHECANDO TODAS EXPECTATIVAS

- `verifyNoMoreInteractions()` – Indica que as únicas interações esperadas para o mock já foram verificadas
- `verifyZeroInteractions()` – Indica que o mock não deveria ter sido usado

```
@Test
public void testGetById() throws Exception{
    Entidade entidade = getEntidadeValida();

    Mockito.when(persistencia.getById(1L)).thenReturn(entidade);

    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1: " + entidade);
    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1 novamente: " + entidade);
    entidade = negocio.getById(2L);
    System.out.println( "Entidade 2: " + entidade);

    Mockito.verify(persistencia, times(2)).getById(1L);
    Mockito.verify(persistencia).getById(2L);
    Mockito.verify(persistencia, never()).getById(3L);
    verifyNoMoreInteractions(persistencia);
}
```

QUESTÃO

- Esse código vai funcionar?

```
@Test
public void testGetById() throws Exception{
    Entidade entidade = getEntidadeValida();


    Mockito.when(persistencia.getById(1L)).thenReturn(entidade);

    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1: " + entidade);
    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1 novamente: " + entidade);
    entidade = negocio.getById(2L);
    System.out.println( "Entidade 2: " + entidade);

    Mockito.verify(persistencia, times(2)).getById(1L);
    verifyNoMoreInteractions(persistencia);
}
```

RESPOSTA

Failure Trace

 org.mockito.exceptions.verifiication.NoInteractionsWanted:

No interactions wanted here:

-> at br.treinamento.EntidadeNegocioMockitoTest.testGetByld(EntidadeNegocioMockitoTest.java:63)

But found this interaction:

-> at br.treinamento.EntidadeNegocio.getByld(EntidadeNegocio.java:77)

For your reference, here is the list of all invocations ([?] - means unverified).

1. -> at br.treinamento.EntidadeNegocio.getByld(EntidadeNegocio.java:77)

2. -> at br.treinamento.EntidadeNegocio.getByld(EntidadeNegocio.java:77)

3. [?]-> at br.treinamento.EntidadeNegocio.getByld(EntidadeNegocio.java:77)

at br.treinamento.EntidadeNegocioMockitoTest.testGetByld(EntidadeNegocioMockitoTest.java:63)

RESET

- Apaga todas as interações realizadas com o mock

```
@BeforeClass
public static void classSetUp() {
    negocio = new EntidadeNegocio();
    persistencia = Mockito.mock(EntidadeDAOInterface.class);
    negocio.setPersistencia(persistencia);
}

@Before
public void methodSetUp() {
    Mockito.reset(persistencia);
}
```

MATCHERS

- Mais flexibilidade nas checagens, permitindo uma generalização
- Pode ser utilizado tanto na expectativa quando na verificação.

```
Mockito.when(persistencia.getById(anyLong())) .thenReturn(entidade);  
Mockito.verify(persistencia, atLeastOnce()).getById(anyLong());
```

- Matchers:
 - anyLong, anyInt, anyObject, etc...

CHAMAR MÉTODO REAL

- Permite chamar o método real de uma CLASSE mockada
 - Não funciona com interface

```
@Test
public void testSalvar() throws Exception{
    Entidade entidade = getEntidadeValida();
    Entidade entidadeRetornada = getEntidadeValida();
    entidadeRetornada.setId(1L);

    when(persistenciaMock.salvar(entidade)).thenReturn(entidadeRetornada);
    doCallRealMethod().when(persistenciaMock).verificarUnicidadeNome(entidadeRetornada);

    Entidade entidadePersistida = negocio.salvar(entidade);
    assertNotNull(entidadePersistida.getId());

    verify(persistenciaMock).salvar(entidade);
}
```

EXPECTATIVAS EM MÉTODOS VOID

- `doNothing()` – ...
- `doThrow(ex)` – Lança a exceção especificada

```
@Test
public void testExcluir() throws Exception{
    Entidade entidade = getEntidadeValida();
    entidade.setTipoDocumento(2);

    doNothing().when(persistencia).excluir(entidade);

    negocio.excluir(entidade);
}

@Test(expected=Exception.class)
public void testExcluirComException() throws Exception{
    Entidade entidade = getEntidadeValida();
    entidade.setTipoDocumento(2);

    doThrow(new Exception())
        .when(persistencia).excluir(entidade);

    negocio.excluir(entidade);
}
```

ORDEM

```
@Test
public void testGetById() throws Exception{
    Entidade entidade = getEntidadeValida();

    Mockito.when(persistencia.getById(1L)).thenReturn(entidade);
    Mockito.when(persistencia.getById(2L)).thenReturn(entidade);

    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1: " + entidade);
    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1: " + entidade);
    entidade = negocio.getById(2L);
    System.out.println( "Entidade 2: " + entidade);

    InOrder inOrder = inOrder(persistencia);
    inOrder.verify(persistencia, times(2)).getById(1L);
    inOrder.verify(persistencia).getById(2L);
}
```


QUESTÃO

- Esse método funciona?

```
@Test
public void testGetById() throws Exception{
    Entidade entidade = getEntidadeValida();

    Mockito.when(persistencia.getById(1L)).thenReturn(entidade);
    Mockito.when(persistencia.getById(2L)).thenReturn(entidade);

    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1: " + entidade);
    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1: " + entidade);
    entidade = negocio.getById(2L);
    System.out.println( "Entidade 2: " + entidade);
    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1: " + entidade);

    InOrder inOrder = inOrder(persistencia);
    inOrder.verify(persistencia, atLeastOnce()).getById(1L);
    inOrder.verify(persistencia).getById(2L);
}
```

CORREÇÃO

```
@Test
public void testGetById() throws Exception{
    Entidade entidade = getEntidadeValida();

    Mockito.when(persistencia.getById(1L)).thenReturn(entidade);
    Mockito.when(persistencia.getById(2L)).thenReturn(entidade);

    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1: " + entidade);
    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1: " + entidade);
    entidade = negocio.getById(2L);
    System.out.println( "Entidade 2: " + entidade);
    entidade = negocio.getById(1L);
    System.out.println( "Entidade 1: " + entidade);

    InOrder inOrder = inOrder(persistencia);
    inOrder.verify(persistencia, times(2)).getById(1L);
    inOrder.verify(persistencia).getById(2L);
    inOrder.verify(persistencia).getById(1L);
}
```

VÁRIOS RETORNOS

```
Mockito.when(persistencia.getById(anyLong()))  
    .thenReturn(entidade1)  
    .thenReturn(entidade2)  
    .thenReturn(entidade3)  
    .thenThrow(new Exception());
```

```
Mockito.when(persistencia.getById(anyLong()))  
    .thenReturn(entidade1, entidade2, entidade3, null)  
    .thenThrow(new Exception());
```

ANNOTATIONS

```
@Spy @InjectMocks private EntidadeNegocioComInterface negocio;  
@Mock private EntidadeDAOInterface persistenciaMock;  
  
@Before  
public void methodSetUp() {  
    MockitoAnnotations.initMocks(this);  
}
```

@MOCK

- Permite criação de mocks mais simples
- A classe de teste fica mais legível
- Necessita do comando *MockitoAnnotations.initMocks(this)* para que os mocks sejam criados.

@INJECTMOCKS

- Minimiza a criação e injeção de spys e mocks
- Os Mocks são encontrados pelo tipo, caso existam mocks do mesmo tipo na classe, informe o atributo “name” na definição do mock

```
@Mock(name = "persistencia")  
private EntidadeDAOInterface persistenciaMock;
```

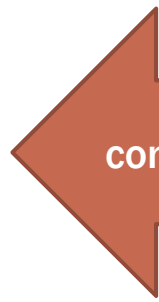
SPY

- É uma forma de mockar parcialmente uma CLASSE
- Em uma classe com spy, o método real será invocado, a menos que este método esteja mockado

```
@Test
public void testAlterarComNomesDiferentes() throws Exception{
    Entidade entidade = getEntidadeValida();
    entidade.setId(1L);
    entidade.setNome("Maria");
    Entidade entidadeBanco = getEntidadeValida();
    entidadeBanco.setId(1L);
    entidadeBanco.setNome("Isabela");

    //when(negocio.getById(1L)).thenReturn(entidadeBanco);
    doReturn(entidadeBanco).when(negocio.getById(1L));

    try{
        negocio.alterar(entidade);
        fail("Exceção deveria ter sido lançada");
    } catch(Exception e) {
        assertEquals("Não é possível alterar o nome da entidade", e.getMessage());
    }
}
```



Mockando
comportamento da classe
de negócio



POWERMOCK

POWERMOCK

- Framework que estende alguns frameworks de mock: EasyMock e Mockito
- Supre as necessidades de mockar construtores, métodos estáticos e finais, usando um ClassLoader modificado.
- Simplifica o uso de Reflection especialmente úteis para criação de testes



INSTALAÇÃO

- Baixar a lib do powermock em:
<http://code.google.com/p/powermock/>
- Para mockito:
 - Adicionar a lib powermock-mockito-<versão>-full.jar

PREPARANDO CLASSE PARA USO

```
@RunWith(PowerMockRunner.class)
@PrepareForTest(EntidadeNegocioComInterface.class)
public class EntidadeNegocioPowerMockitoTest {

    private EntidadeNegocioComInterface negocio;
    private EntidadeDAOInterface persistenciaMock;

    @Before
    public void methodSetUp() {
        negocio = PowerMockito.spy(new EntidadeNegocioComInterface());
        persistenciaMock = PowerMockito.mock(EntidadeDAOInterface.class);
        negocio.setPersistencia(persistenciaMock);
    }
}
```

MOCKANDO MÉTODOS PRIVADOS

- Vamos supor que “validarCamposObrigatórios” é muito complexa e não desejamos se preocupar com o comportamento dela nesse teste...

```
@Test
public void testSalvarSemValidarCamposObrigatorios() throws Exception{
    Entidade entidade = getEntidadeValida();
    Entidade entidadeRetornada = getEntidadeValida();
    entidadeRetornada.setId(1L);

    Mockito.doNothing().when(negocio, "validarCamposObrigatorios", entidade);
    when(persistenciaMock.salvar(entidade)).thenReturn(entidadeRetornada);
    when(persistenciaMock.verificarUnicidadeNome(entidadeRetornada)).thenReturn(true);

    Entidade entidadePersistida = negocio.salvar(entidade);
    assertNotNull(entidadePersistida.getId());

    verify(persistenciaMock).salvar(entidade);
    verify(persistenciaMock).verificarUnicidadeNome(entidadeRetornada);
    Mockito.verifyPrivate(negocio).invoke("validarCamposObrigatorios", entidade);
}
```

TESTANDO MÉTODOS PRIVADOS

```
@Test
public void testValidarCamposObrigatorios() throws Exception{
    Entidade entidade = getEntidadeValida();
    entidade.setNome(null);

    try {
        Whitebox.invokeMethod(negocio, "validarCamposObrigatorios", entidade);
        fail("Exceção deveria ter sido lançada");
    } catch (Exception e) {
        assertEquals("O nome é obrigatório", e.getMessage());
    }
}
```

MOCKANDO CONSTRUTORES

- A classe que está criando a entidade deve estar no `@PrepareForTest`
- O método `getEntidadeVazia()` instancia uma nova entidade e retorna a mesma.

```
@Test
public void testConstrutor() throws Exception{
    Entidade ent = getEntidadeValida();
    Mockito.whenNew(Entidade.class).withNoArguments().thenReturn(ent);
    Entidade entidade = negocio.getEntidadeVazia();
    assertEquals("Entidade ABC", entidade.getNome());

    Mockito.verifyNew(Entidade.class).withNoArguments();
}
```

QUESTÃO

■ O que acontece com esse código?

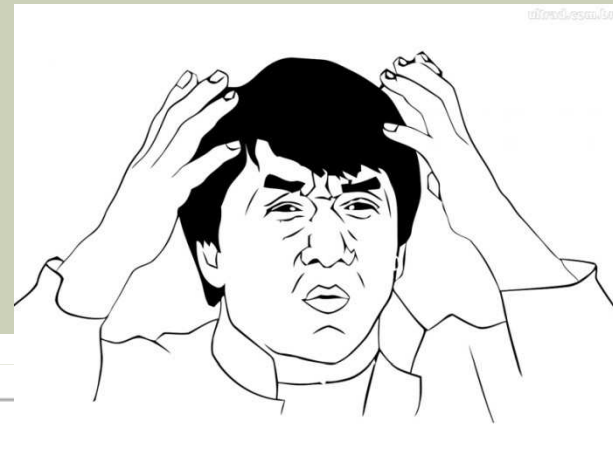
■ Classe de Negócio

```
public boolean validarData() {  
    Calendar calendar = Calendar.getInstance();  
    return calendar.get(Calendar.MONTH) == Calendar.JANUARY;  
}
```


■ Classe de Teste

```
@Test  
public void testValidarDataMesJaneiro() {  
    Calendar calendar = Calendar.getInstance();  
    calendar.set(Calendar.MONTH, Calendar.JANUARY);  
  
    when(Calendar.getInstance()).thenReturn(calendar);  
  
    assertTrue(negocio.validarData());  
}
```

RESPOSTA



Failure Trace

 org.mockito.exceptions.misusing.MissingMethodInvocationException:
when() requires an argument which has to be 'a method call on a mock'.


For example:

```
when(mock.getArticles()).thenReturn(articles);
```



Also, this error might show up because:

1. you stub either of: final/private/equals()/hashCode() methods.
Those methods *cannot* be stubbed/verified.
2. inside when() you don't call method on mock but on some other object.

 at br.treinamento.EntidadeNegocioMockitoTest.testValidarDataJaneiro(EntidadeNegocioMockitoTest.java:125)

MOCKANDO MÉTODOS ESTÁTICOS

```
@Test
public void testValidarDataMesJaneiro() {
    Calendar calendar = Calendar.getInstance();
    calendar.set(Calendar.MONTH, Calendar.JANUARY);

    Mockito.mockStatic(Calendar.class);
    when(Calendar.getInstance()).thenReturn(calendar);

    assertTrue(negocio.validarData());
    assertTrue(negocio.validarData());

    Mockito.verifyStatic(times(2));
    Calendar.getInstance();
}
```

THE DARK POWERS OF POWERMOCK



- O PowerMock deve ser usado apenas onde o mesmo seja estritamente necessário.
- PowerMock e EcEmma não se dão bem

COMO ESCAPAR DO POWERMOCK?

- Testar métodos privados
 - Métodos privados podem ser acessados via métodos públicos
- Testar métodos estáticos
 - Refatorando a classe...

```
public boolean validarData() {  
    Calendar calendar = getDataAtual();  
    return calendar.get(Calendar.MONTH) == Calendar.JANUARY;  
}  
  
public Calendar getDataAtual() {  
    return Calendar.getInstance();  
}
```

- Com o próprio mockito é possível configurar o comportamento

```
doReturn(calendar).when(negocio.getDataAtual());
```

PAGE OBJECT PATTERN

FORMAS DE CRIAR TESTES COM SELENIUM

- IDE Dump pattern
- Métodos funcionais e modulares
- Page Object

PAGE OBJECT

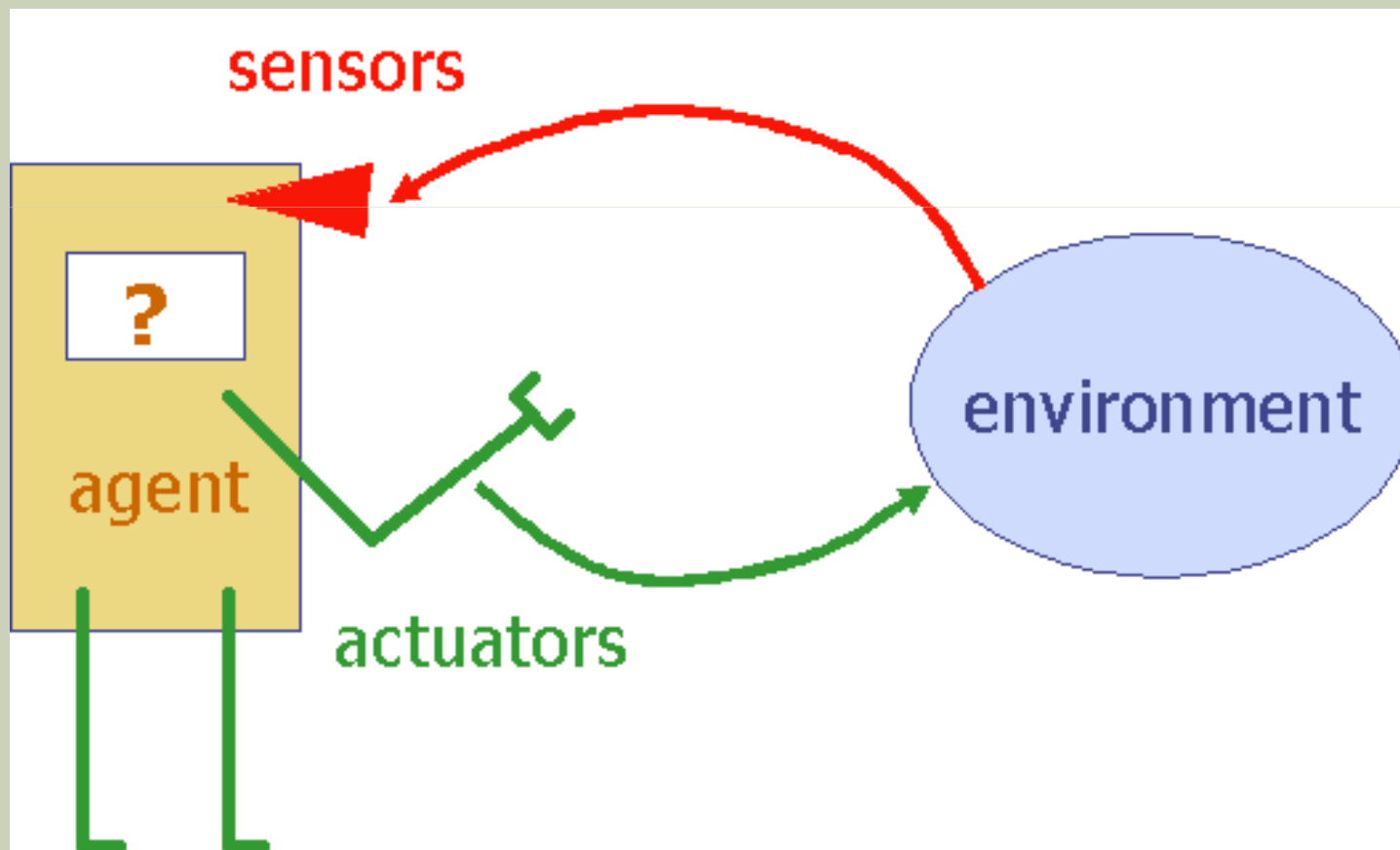
- A classe de teste se divide em duas: Test e Page
- Classe Test: Responsável por toda a lógica e assertivas durante os testes [TestClass]
- Classe Page: Responsável por realizar os acessos às páginas. Sejam elas para coletar informações ou para executar ações. [PageClass]

PARA QUE SERVE?

- Expõe métodos que refletem coisas que podemos VER e FAZER na página
 - `getNomeUsuarioLogado`
 - `salvarFormulario`
- Esconde detalhes de como “dizer” para o browser como encontrar o que desejamos

COMPLICADO?

- Vamos apelar para a Inteligência Artificial, lembram do agente?



MELHORA VENDENDO O CÓDIGO?

```
page.acessarTelaBuscaCEP();  
verificarTelaBuscaCEP();  
  
page.setLogradouro("Rua professor otavio lobo");  
page.setTipoCep("Localidade/Logradouro");  
page.setSemelhantes(true);  
page.clicarBotaoBuscar();
```



Classe Test

```
public void acessarTelaBuscaCEP() {  
    driver.get("http://www.correios.com.br/");  
    clicarLink("Busca CEP");  
}  
  
public void setLogradouro(String valor) {  
    escrever(By.name("relaxation"), valor);  
}  
  
public void setTipoCep(String valor) {  
    selecionarCombo(By.name("TipoCep"), valor);  
}  
  
public void setSemelhantes(boolean b) {  
    clicarElemento(By.xpath("//input[@name='semelhante' and @value='" + (b? "S" : "N") + "']"));  
}
```



Classe Page

VANTAGENS

- Testes mais legíveis.
- Possibilita reuso.
- Facilita manutenção dos elementos.
- Permite a troca de frameworks sem grandes alterações na classe Test
 - Em teoria, a classe page que deverá ser alterada

PAGE FACTORY

- Implementação do Selenium para “facilitar” a utilização do PageObject Pattern
- @FindBy()
 - Cria um lazy proxy para um determinado WebElement
- @CacheLookup
 - Evita vários lookups para o mesmo elemento.
- Necessita fazer um `PageFactory.initElements(driver, this);`

CLASSE PAGE

```
@FindBy(css = "div.tituloimagem > h1")
private WebElement campoTitulo;

@FindBy(name = "relaxation")
private WebElement logradouro;

@FindBy(name = "TipoCep")
private WebElement tipoCep;

@FindBy(xpath = "//button[@type='submit']")
private WebElement botaoBuscar;

public BuscaCEPPage(WebDriver driver) {
    super(driver);
    PageFactory.initElements(driver, this);
}
```

Domínio de Navegação

```
public void setLogradouro(String valor){
    escrever(logradouro, valor);
}

public ResultadoBuscaCEPPage clicarBotaoBuscar(){
    clicarElemento(botaoBuscar);
    return new ResultadoBuscaCEPPage(driver);
}
```

CLASSE DE TESTE

```
@Test
public void testCEPExistente() {
    MainPage mainPage = new MainPage(driver);
    assertTrue(mainPage.isPaginaCorreta());

    BuscaCEPPage buscaCEPPage = mainPage.clicarLinkBuscaCEP();
    assertTrue(buscaCEPPage.isPaginaCorreta());

    buscaCEPPage.setLogradouro("Rua professor otavio lobo");
    buscaCEPPage.setTipoCep("Localidade/Logradouro");
    buscaCEPPage.setSemelhantes(true);
    ResultadoBuscaCEPPage resultadoBuscaCEPPage =
        buscaCEPPage.clicarBotaoBuscar();
    assertTrue(resultadoBuscaCEPPage.isPaginaCorreta());

    assertEquals("Rua Professor Otávio Lobo - até 449/450",
        resultadoBuscaCEPPage.obterLogradouroPorIndex(1));
    assertEquals("Rua Professor Otávio Lobo - de 451/452 ao fi",
        resultadoBuscaCEPPage.obterLogradouroPorIndex(2));
    DetalheLogradouroBuscaCEPPage detalhePage =
        resultadoBuscaCEPPage.clicarLogradouroPorIndex(2);
    assertTrue(detalhePage.isPaginaCorreta());

    Assert.assertEquals("Cocó", detalhePage.getBairro());
}
```

MÃOS À OBRA

- Refatorar os testes para se adequar ao padrão [Page/Test]