

# Testes em aplicações Java EE com Arquillian

Paulo Jerônimo

---

# Table of Contents

|   |    |
|---|----|
| 1. Introdução .....   | 1  |
| 2. O que você precisa saber .....   | 2  |
| 3. Instalando e configurando o Fedora .....   | 3  |
| 3.1. Configurando o arquivo /etc/sudoers .....  | 3  |
| 3.2. Trabalhando com o Fedora mais atual .....  | 3  |
| 3.3. Instalando o Oracle JDK .....  | 4  |
| 3.4. Instalando e configurando outros softwares proprietários .....   | 5  |
| 4. Montando um ambiente para este tutorial .....  | 6  |
| 4.1. Baixe o JBoss EAP .....  | 6  |
| 4.2. Instalando e configurando um ambiente para a construção, execução e testes de aplicações Java EE ..... | 6  |
| 4.2.1. Observando a estrutura construída .....  | 6  |
| 4.2.2. Iniciando e parando o JBoss .....  | 7  |
| 4.2.3. Iniciando o Eclipse e instalando o JBoss Developer Studio .....                                      | 7  |
| 5. Um revisão da teoria (na prática) sobre TDD e BDD .....  | 9  |
| 5.1. TDD sem JUnit, para os bravos .....  | 9  |
| 5.2. BDD, com Cucumber .....  | 17 |
| 6. Testes reais com o Arquillian .....  | 24 |
| 6.1. TDD com Arquillian: desenvolvendo uma aplicação Java EE do zero .....                                  | 24 |
| 6.2. Executando e testando a aplicação tasks-rs .....   | 24 |
| 6.2.1. Iniciando o JBoss e implantando a aplicação .....  | 24 |
| 6.2.2. Testando a aplicação .....   | 24 |
| 6.3. Executando e testando a aplicação kitchensink-angularjs .....  | 25 |
| 7. Test-Driven JavaScript Development .....   | 26 |
| 8. Referências .....  | 27 |

---

# Chapter 1. Introdução

Este documento (última versão em <http://paulojeronimo.github.io/javaee-tutorial-testes/> e também [disponível em PDF<sup>1</sup>](#)) é um tutorial prático que apresenta conceitos e exemplos da aplicação de [Test Driven Development<sup>2</sup>](#) (TDD) ou [Behavior Driven Development<sup>3</sup>](#) (BDD) no desenvolvimento de aplicações Java EE. Ele é útil para o início de um estudo mais aprofundado sobre o framework [Arquillian<sup>4</sup>](#) na produção de aplicações que integram soluções Java EE com frameworks como o [AngularJS<sup>5</sup>](#).

Este material utiliza [scripts para que automatizam a montagem de um ambiente de desenvolvimento para aplicações Java EE<sup>6</sup>](#) e, embora não seja o foco prioritário deste material, ele também apresenta algumas soluções e exemplos sobre como realizar testes em aplicações Javascript, em especial as desenvolvidas com o AngularJS no `frontend` e Java EE no `backend`.

O conteúdo deste material foi projetado para ser apresentado, presencialmente por um instrutor, em até oito horas. Contudo, o estudo completo das [referências](#) citadas neste documento extrapola, e muito, esta carga horária. Portanto, fica cargo do estudante correr atrás de expandir seus conhecimentos através do aprofundamento do estudo dessas referências.

---

<sup>1</sup> [./javaee-tutorial-testes.pdf](#)

<sup>2</sup> [http://pt.wikipedia.org/wiki/Test\\_Driven\\_Development](http://pt.wikipedia.org/wiki/Test_Driven_Development)

<sup>3</sup> [http://pt.wikipedia.org/wiki/Behavior\\_Driven\\_Development](http://pt.wikipedia.org/wiki/Behavior_Driven_Development)

<sup>4</sup> [http://arquillian.org/guides/getting\\_started\\_pt/](http://arquillian.org/guides/getting_started_pt/)

<sup>5</sup> <https://angularjs.org/>

<sup>6</sup> <https://github.com/paulojeronimo/javaee-ambiente>

---

## Chapter 2. O que você precisa saber

Os roteiros deste material foram produzidos para a execução, inicialmente, num ambiente Linux ([Fedora 21](http://fedoraproject.org)<sup>1</sup>). Seu desenvolvimento foi realizado com base na instalação padrão de uma `workstation` nessa versão do Fedora. Contudo, esses roteiros também podem ser facilmente adaptados para a execução em outras distribuições Linux (como RHEL, CentOS, Debian ou Ubuntu) assim como no OS X e no Windows (num shell Cygwin). Para executar as tarefas deste material, espera-se que o estudante tenha facilidade para compreender e executar scripts no Bash.

Os exemplos de código apresentados neste material são, primariamente, escritos em Java e JavaScript. Obviamente, espera-se que o estudante já tenha experiência nessas linguagens. Da mesma forma, o estudo detalhado das tecnologias testadas nos exemplos discutidos neste material está fora de seu escopo. Portanto, espera-se que o estudante já possua conhecimentos nas APIs e frameworks Java/Javascript que são alvo dos códigos de teste (exemplos: JPA, EJB, JAX-RS, AngularJS, etc).

Espera-se que o estudante, ao iniciar a leitura deste material, já possua algum embasamento em [JUnit](http://junit.org)<sup>2</sup> e em TDD. Caso este não seja o caso, é recomendado que o estudante faça a leitura das referências básicas (marcadas como "*Ref. básica*") apresentadas ao final deste documento.

---

<sup>1</sup> <http://fedoraproject.org>

<sup>2</sup> <http://junit.org>

---

# Chapter 3. Instalando e configurando o Fedora

Está fora do escopo deste material apresentar um passo a passo completo para a instalação do Fedora. Na Internet há vários materiais a respeito disso. Entretanto, após instalado, o estudante deverá se certificar de fazer configurações e instalações de pacotes recomendadas neste tópico.

## 3.1. Configurando o arquivo `/etc/sudoers`

Para que a execução de scripts dependentes do uso do comando `sudo` não fique solicitando uma senha, é preciso que o arquivo `/etc/sudoers` contenha linhas configuradas conforme a saída do comando abaixo:

```
.....  
$ sudo grep wheel /etc/sudoers  
## Allows people in group wheel to run all commands  
#%wheel    ALL=(ALL)    ALL  
%wheel    ALL=(ALL)    NOPASSWD: ALL  
.....
```

Para editar esse arquivo, deixando-o apresentável conforme a saída acima, execute:

```
.....  
sudo visudo  
.....
```

## 3.2. Trabalhando com o Fedora mais atual

Todos os roteiros deste tutorial foram desenvolvidos com a versão mais atual dos pacotes do Fedora. Então, para ter estas mesmas versões em teu ambiente, execute o `update` dos pacotes:

```
.....  
sudo yum -y update  
.....
```

Em seguida, reinicie o Fedora (para termos certeza que estaremos utilizando o último kernel disponível):

```
.....  
sudo shutdown -r now  
.....
```

Após a reinicialização da máquina, observe a seleção (automática e configurada no grub) da última versão do kernel, no momento do boot.

Logue-se com teu usuário e para manter apenas o kernel mais novo na lista de opções do grub, execute:

```
sudo package-cleanup -y --oldkernels --count=1
```

### 3.3. Instalando o Oracle JDK

Alguns sites, como os do [Banco do Brasil](http://www.bb.com.br)<sup>1</sup> e do [Itaú](http://www.itau.com.br)<sup>2</sup>, dependem da instalação do [Oracle JRE](http://www.oracle.com/technetwork/java/javase/downloads/index.html)<sup>3</sup>. Então, é interessante ter esta JRE instalada. E, apesar do OpenJDK fazer parte da instalação padrão do Fedora 21, utilizaremos o [Oracle JDK](http://www.oracle.com/technetwork/java/javase/downloads/index.html)<sup>4</sup>, neste material.

Para instalar o Oracle JRE (e JDK) utilizaremos o [Fedy](https://satya164.github.io/fedy/fedy-installer)<sup>5</sup>, executando os comandos a seguir:

```
curl -sSL https://satya164.github.io/fedy/fedy-installer | sudo bash
sudo fedy -e oracle_jre oracle_jdk
```

Em seguida, configuraremos os binários que serão executados do Java, utilizando o comando `alternatives`:

```
sudo alternatives --install /usr/bin/java java /usr/java/latest/jre/bin/
java 200000
sudo alternatives --install /usr/bin/javaws javaws /usr/java/latest/jre/
bin/javaws 200000
sudo alternatives --install /usr/lib64/mozilla/plugins/libjavaplugin.so
libjavaplugin.so.x86_64 /usr/java/latest/jre/lib/amd64/libnpjp2.so 200000
sudo alternatives --install /usr/bin/javac javac /usr/java/latest/bin/
javac 200000
sudo alternatives --install /usr/bin/jar jar /usr/java/latest/bin/
jar 200000

sudo alternatives --config java
```

---

<sup>1</sup> <http://www.bb.com.br>

<sup>2</sup> <http://www.itau.com.br>

<sup>3</sup> <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

<sup>4</sup> <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

<sup>5</sup> <https://satya164.github.io/fedy/>

```
sudo alternatives --config javaws
sudo alternatives --config libjavaplugin.so.x86_64
sudo alternatives --config javac
sudo alternatives --config jar
```

---

Pronto, agora testemos a execução de applets Java acessando a página "[Verificar Versão do Java](#)"<sup>6</sup> e também os sites dos bancos brasileiros.

### 3.4. Instalando e configurando outros softwares proprietários

Particularmente, eu utilizo o Fedy para que a ele realize algumas configurações opcionais no Fedora e também a instale alguns softwares proprietários.

Você pode obter a lista de configurações e instalações de softwares que o Fedy pode fazer através de sua interface gráfica. Alternativamente, pela linha de comando, você também pode obter esta listagem:

---

```
sudo fedy -e list
```

---

Para fazer minhas configurações e instalações através do Fedy eu executo o seguinte comando:

---

```
sudo fedy -e adobe_flash core_fonts dvd_playback essential_soft
font_rendering google_chrome google_talkplugin media_codecs
nautilus_dropbox rpmfusion_repos skype_linux teamviewer_linux
```

---

---

<sup>6</sup> [https://www.java.com/pt\\_BR/download/installed.jsp](https://www.java.com/pt_BR/download/installed.jsp)

---

# Chapter 4. Montando um ambiente para este tutorial

## 4.1. Baixe o JBoss EAP

Clique nos links abaixo para fazer o download de alguns arquivos que utilizaremos (será solicitado um login):

- <http://www.jboss.org/download-manager/file/jboss-eap-6.3.0.GA.zip>
- <http://www.jboss.org/download-manager/file/jboss-eap-6.3.0.GA-quickstarts.zip>

Crie o diretório que conterá os arquivos baixados e copie-os para ele:

```
mkdir -p ~/javaee-tutorial-testes.backup/javaee-ambiente.instaladores
cp ~/Downloads/jboss-eap-6.3.0.* !$
```

## 4.2. Instalando e configurando um ambiente para a construção, execução e testes de aplicações Java EE

Baixe e execute os scripts de montagem de ambiente através dos seguintes comandos:

```
cd && git clone http://github.com/paulojeronimo/javaee-tutorial-testes
cd javaee-tutorial-testes
cp config.exemplo config
vim config
./instalar
```

O script `instalar` criará o usuário `javaee` e instalará um ambiente completo no `$HOME` desse usuário para que você possa realizar as tarefas apresentadas neste documento. Assim que terminada a instalação, você precisará se tornar este usuário para executar quaisquer tarefas.

### 4.2.1. Observando a estrutura construída

Logue-se como usuário `javaee`:



```
sudo su - javaee
```

Observe a estrutura de diretórios/arquivos montada no \$HOME deste usuário:

```
tree -L 1
```

## 4.2.2. Iniciando e parando o JBoss

Para iniciar o JBoss, execute:

```
jboss_start
```

Para observar os logs do JBoss em execução, execute:

```
jboss_tail &
```



Isso fará com que qualquer alteração no log do JBoss seja apresentada no shell corrente. Para encerrar esta apresentação, a qualquer momento, execute:

```
pkill tail
```

Para parar a execução do JBoss, execute:

```
jboss_stop
```

Para reinstalar o JBoss (em alguns exemplos faremos isto), execute:

```
jboss_instalar
```

## 4.2.3. Iniciando o Eclipse e instalando o JBoss Developer Studio

Para iniciar o Eclipse, execute:

```
eclipse &> /dev/null &
```

Para instalar o [JBoss Developer Studio<sup>1</sup>](#), siga os passos descritos em [na página do produto<sup>2</sup>](#). Alternativamente, se ao invés de utilizar o procedimento de instalação descrito nesta página você desejar fazer a instalação offline, siga os passos descritos a seguir.

Baixe o zip com o update site do JBoss Developer Studio através do script a seguir:

```
.....  
jbdevstudio_baixar  
.....
```

Terminado o download, o arquivo baixado será salvo no diretório `~/instaladores`). Acesse a opção de menu `Help > Install New Software... > Add... > Archive...`, selecione esse arquivo e prossiga com a instalação.

Para salvar o Eclipse configurado com os plugins que você instalou, encerre sua execução e execute:

```
.....  
eclipse_salvar  
salvar_instaladores  
.....
```

---

<sup>1</sup> <http://tools.jboss.org/downloads/devstudio/index.html>

<sup>2</sup> <http://tools.jboss.org/downloads/devstudio/luna/8.0.2.GA.html>

---

# Chapter 5. Um revisão da teoria (na prática) sobre TDD e BDD

Talvez você queira dar uma olhada numa [apresentação que fiz para a Capes, em 2013<sup>1</sup>](#).

## 5.1. TDD sem JUnit, para os bravos

Leia o tutorial [Test-Driven Development \(TDD\) em Java \(Parte 1\)<sup>2</sup>](#) mas, não execute-o.

Agora, você irá executá-lo de uma maneira ainda mais passo a passo e simples. Apenas brincando de copiar e colar os comandos, a seguir, num shell sendo executado pelo usuário `javaee`. Dessa forma, você colocará TDD em prática e sem a preocupação de utilizar qualquer IDE.

Crie o diretório `~/exemplos/tdd` e vá para ele:

```
cd && mkdir -p exemplos/tdd
cd !$
```

Crie a classe `MatematicaTest`:

```
cat > MatematicaTest.java <<EOF
public class MatematicaTest {
    public void testFatorial() {
    }

    public void testFibonacci() {
    }

    public static void main(String args[]) {
        MatematicaTest mt = new MatematicaTest();
        try {
            mt.testFatorial();
            System.out.println("testFatorial() passou!");
            mt.testFibonacci();
            System.out.println("testFibonacci() passou!");
        } catch (AssertionFailedError e) {
```

---

<sup>1</sup> <http://a.paulojeronimo.info/capes/processo-de-testes/index.html>

<sup>2</sup> <https://dl.dropboxusercontent.com/u/345266/artigos/tdd/1/index.html>

```
        System.out.println("Teste falhou:");
        e.printStackTrace();
    } catch (Exception e) {
        System.out.println("Teste provocou exceção:");
        e.printStackTrace();
    }
}
}
EOF
```

---

Compile o código e verifique que dá erro.

---

```
javac MatematicaTest.java
```

---

Conserte o erro e recompile o código:

---

```
cat > AssertionError.java <<EOF
public class AssertionError extends Error {
    public AssertionError(String message) {
        super(message);
    }
}
EOF
!-2
```

---

Percebeu que você acabou de criar um mini **framework** de testes (JUnit)!? =)  
Agora, comece a implementar os métodos de testes para, em seguida, criar a implementação que fará estes testes passarem.

Modifique a classe **MatematicaTest** implementando o método **testFatorial**:

---

```
patch MatematicaTest.java << EOF
--- MatematicaTest.java.1    2015-02-08 18:15:02.007920683  -0200
+++ MatematicaTest.java      2015-02-08 18:27:09.016219866  -0200
@@ -1,10 +1,27 @@
 public class MatematicaTest {
+ public static void fail(String message) {
+     throw new AssertionError(message);
+ }
+
+ public void testFatorial() {
+     testFatorialComArgumentoNegativo();
+     //testFatorialDe0();
+     //testFatorialDe5a7();
 }
```

```
    }

    public void testFibonacci() {
    }

+   public void testFatorialComArgumentoNegativo() {
+       long result = -1;
+       try {
+           result = Matematica.fatorial(-1);
+           fail("fatorial(-1) deveria ter lançado
+   IllegalArgumentException");
+       } catch (IllegalArgumentException e) {
+           // ok, isto era esperado!
+       }
+   }
+
+   public static void main(String args[]) {
+       MatematicaTest mt = new MatematicaTest();
+       try {
EOF
```

Observe as alterações, compile e verifique que dá erro pois, agora, será necessário criar a classe `Matematica` que implementará o método fatorial.

---

```
vim MatematicaTest.java
```

---

Dentro do vim, pressione um **Ctrl+Z** para voltar ao shell e, em seguida, compile o código:

---

```
javac MatematicaTest.java
```

---



A qualquer momento você pode retornar ao Vim, a partir do shell, executando o comando `fg`.

Crie a classe `Matematica`, com uma implementação que fará o método de testes passar e, em seguida, recompile e reexecute a classes de testes:

---

```
cat > Matematica.java <<EOF
public class Matematica {
    public static long fatorial(long n) {
        if (n < 0)
            throw new IllegalArgumentException();
    }
}
```

```
        return 0;
    }
}
EOF
javac MatematicaTest.java
java MatematicaTest
```

---

Observe que o teste passou! \o/ Mas, ainda faltam vários testes e implementações a realizar até que você chegue ao código final. Siga em frente, criando um teste para validar o fatorial de 0. Em seguida, compile e reexecute. Você notará que sua implementação para a classe `Matematica` precisará de mudanças em função do novo teste.

---

```
patch MatematicaTest.java <<EOF
--- MatematicaTest.java.2    2015-02-08 18:27:38.001992577 -0200
+++ MatematicaTest.java 2015-02-08 18:31:41.453083559 -0200
@@ -3,9 +3,17 @@
         throw new AssertionError(message);
     }

+    public static void assertEquals(String message, long expected, long
+    actual) {
+        if (expected != actual) {
+            throw new AssertionError(message +
+            "\nValor esperado: " + expected +
+            "\nValor obtido: " + actual);
+        }
+    }
+
     public void testFatorial() {
         testFatorialComArgumentoNegativo();
-        //testFatorialDe0();
+        testFatorialDe0();
+        //testFatorialDe5a7();
     }

@@ -22,6 +30,10 @@
     }

+    public void testFatorialDe0() {
+        assertEquals("fatorial(0) != 1", 1, Matematica.fatorial(0));
+    }
+
     public static void main(String args[]) {
```

---

```
MatematicaTest mt = new MatematicaTest();
try {
```

EOF

```
javac MatematicaTest.java
java MatematicaTest
```

---

Este deverá ser o erro apresentado na execução do último comando:

---

```
Teste falhou:
AssertionFailedError: fatorial(0) != 1
Valor esperado: 1
Valor obtido: 0
    at MatematicaTest.assertEquals(MatematicaTest.java:8)
    at MatematicaTest.testFatorialDe0(MatematicaTest.java:34)
    at MatematicaTest.testFatorial(MatematicaTest.java:16)
    at MatematicaTest.main(MatematicaTest.java:40)
```

---

Para corrigi-lo, você deverá modificar a implementação do método `fatorial` na classe `Matematica`. Daí você poderá recompilar e fazer o teste passar novamente:

---

```
patch Matematica.java <<EOF
--- Matematica.java.1    2015-02-08 18:39:36.414359163 -0200
+++ Matematica.java      2015-02-08 18:41:59.534234153 -0200
@@ -2,6 +2,8 @@
     public static long fatorial(long n) {
         if (n < 0)
             throw new IllegalArgumentException();
+        if (n == 0)
+            return 1;
         return 0;
     }
 }
EOF
javac *.java
java MatematicaTest
```

---

Implemente o método de teste `testFatorialDe5a7` na classe `MatematicaTest` e, em seguida, faça o teste passar alterando, também, a classe `Matematica`:

---

```
patch MatematicaTest.java <<EOF
--- MatematicaTest.java.3 2015-02-08 18:13:34.544606524 -0200
+++ MatematicaTest.java   2015-02-08 18:55:56.352636333 -0200
```

```
@@ -14,7 +14,7 @@
    public void testFatorial() {
        testFatorialComArgumentoNegativo();
        testFatorialDe0();
-        //testFatorialDe5a7();
+        testFatorialDe5a7();
    }

    public void testFibonacci() {
@@ -34,6 +34,16 @@
        assertEquals("fatorial(0) != 1", 1, Matematica.fatorial(0));
    }

+    public void testFatorialDe5a7() {
+        for (int i = 5; i <= 7; i++) {
+            switch (i) {
+                case 5: assertEquals("fatorial(5) != 120", 120,
Matematica.fatorial(5)); break;
+                case 6: assertEquals("fatorial(6) != 720", 720,
Matematica.fatorial(6)); break;
+                case 7: assertEquals("fatorial(7) != 5040", 5040,
Matematica.fatorial(7)); break;
+            }
+        }
+    }
+
    public static void main(String args[]) {
        MatematicaTest mt = new MatematicaTest();
        try {
EOF
patch Matematica.java <<EOF
--- Matematica.java.2    2015-02-08 18:57:08.081070792 -0200
+++ Matematica.java 2015-02-08 19:06:05.813831088 -0200
@@ -4,6 +4,12 @@
        throw new IllegalArgumentException();
        if (n == 0)
            return 1;
+    else if (n == 5)
+        return 120;
+    else if (n == 6)
+        return 720;
+    else if (n == 7)
+        return 5040;
        return 0;
    }
}
```



EOF

```
javac *.java
java MatematicaTest
```

---

Enfim, implemente o método `testFatorialDeN` na classe `MatematicaTest` e execute-a:

---

```
patch MatematicaTest.java <<EOF
--- MatematicaTest.java.4    2015-02-09 01:58:00.285104599 -0200
+++ MatematicaTest.java 2015-02-09 02:04:24.212655227 -0200
@@ -1,3 +1,5 @@
+import java.util.Random;
+
+  public class MatematicaTest {
+    public static void fail(String message) {
+      throw new AssertionError(message);
@@ -15,6 +17,7 @@
+    testFatorialComArgumentoNegativo();
+    testFatorialDe0();
+    testFatorialDe5a7();
+    testFatorialDeN();
+  }

+    public void testFibonacci() {
@@ -43,6 +46,31 @@
+    }
+  }

+  public void testFatorialDeN() {
+    long result;

+    // testa a regra "fatorial(n) = n * fatorial(n-1)" 30 vezes
+    // n é um número aleatório entre 0 e 20.
+    // Porque 20? Porque este é o inteiro máximo cujo fatorial
+    // não estrapola Long.MAX_VALUE: Veja em FatorialMaximo.java
+    Random r = new Random();
+    int n;
+    for (int i = 0; i < 30; i++) {
+      n = r.nextInt(20 + 1);
+      if (n < 0)
+        assert true : "n nunca deveria ser negativo!";
+      else {
+        result = Matematica.fatorial(n);
```

```
+         System.out.printf("%2d: Fatorial de %2d = %d\n", i, n,
+         result);
+         if (n == 0)
+             assertEquals("fatorial(0) != 1", result, 1);
+         else
+             assertEquals("fatorial("+n+") != "+n+" *
+         fatorial("+n-1+")",
+         result, n * Matematica.fatorial(n-1));
+     }
+ }

    public static void main(String args[]) {
        MatematicaTest mt = new MatematicaTest();
EOF
javac MatematicaTest.java
java MatematicaTest
```

---

Observe que, agora, seu programa de teste sempre irá falhar em algum momento. Não lhe restará outra alternativa a não ser fazer a implementação correta da classe `Matematica`:

---

```
patch Matematica.java <<EOF
--- Matematica.java.3    2015-02-09 01:58:11.897021389 -0200
+++ Matematica.java      2015-02-09 02:14:33.710629599 -0200
@@ -2,14 +2,6 @@
     public static long fatorial(long n) {
         if (n < 0)
             throw new IllegalArgumentException();
-        if (n == 0)
-            return 1;
-        else if (n == 5)
-            return 120;
-        else if (n == 6)
-            return 720;
-        else if (n == 7)
-            return 5040;
-        return 0;
+        return n == 0 ? 1 : n * fatorial(n - 1);
     }
 }
EOF
```

---

Finalmente, seu programa de testes e sua implementação para a classe Matematica estarão corretos. Compile as classes e reexecute o programa de testes várias vezes para ter certeza disso:

```
javac *.java
for i in `seq 4`; do java MatematicaTest | (less; read n); done
```



1. Exercício: agora, utilizando o Eclipse e o JUnit, utilize TDD para implementar o cálculo da série Fibonacci.

## 5.2. BDD, com Cucumber

Leia o artigo [TDD e BDD em Aplicações Java EE com JUnit, Arquillian, Selenium e Cucumber, parte 1](#)<sup>3</sup> mas, não execute-o.

Agora, vamos executá-lo utilizando o ambiente que montamos para o usuário `javaee`:

Comece pela criação da `feature`:

```
d=~/.exemplos/bdd; rm -rf $d && mkdir -p $d && cd $d
d=src/test/resources/com/ladoservidor/cucumber/helloworld; mkdir -p $d
cat > $d/helloworld.feature <<'EOF'
Feature: Hello World

  Scenario: Say hello
    Given I have a hello app with "Hello"
    When I ask it to say hi
    Then it should answer with "Hello World"
EOF
```

Crie o `pom.xml` do projeto:

```
cat > pom.xml <<'EOF'
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
```

---

<sup>3</sup> <http://blog.ladoservidor.com/2013/04/agilebrazil-1.html>

```
<groupId>com.ladoservidor</groupId>
<artifactId>cucumber-jvm-helloworld</artifactId>
<version>1.0</version>
<packaging>jar</packaging>
<name>cucumber-jvm/HelloWorld</name>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.5.1</version>
      <configuration>
        <encoding>UTF-8</encoding>
        <source>1.6</source>
        <target>1.6</target>
      </configuration>
    </plugin>

    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>2.12.2</version>
      <configuration>
        <useFile>false</useFile>
      </configuration>
    </plugin>
  </plugins>
</build>

<dependencies>
  <dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>1.1.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>1.1.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
```

```
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
</dependencies>
</project>
EOF
```

---

Observe a estrutura do projeto, até agora:

```
tree
```

---

Crie a classe `RunCukesTest` que executará os testes do Cucumber através do JUnit:

```
d=src/test/java/com/ladoservidor/cucumber/helloworld; mkdir -p $d
cat > $d/RunCukesTest.java <<'EOF'
package com.ladoservidor.cucumber.helloworld;

import cucumber.api.junit.Cucumber;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@Cucumber.Options(
    format = {
        "pretty",
        "html:target/cucumber-html-report",
        "json-pretty:target/cucumber-json-report.json"
    }
)
public class RunCukesTest {
}
EOF
```

---

Execute o maven:

```
mvn test
```

---

Observe a estrutura gerada para no diretório `target` e abra o arquivo `target/cucumber-html-report/index.html`:

```
tree target
browse target/cucumber-html-report/index.html
```

---

Crie a classe `HelloStepdefs`:

```
cat > $d/HelloStepdefs.java <<'EOF'
package com.ladoservidor.cucumber.helloworld;

import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;

import static org.junit.Assert.assertEquals;

public class HelloStepdefs {
    private Hello hello;
    private String hi;

    @Given("^I have a hello app with \"([^\"]*)\"$")
    public void I_have_a_hello_app_with(String greeting) {
        hello = new Hello(greeting);
    }

    @When("^I ask it to say hi$")
    public void I_ask_it_to_say_hi() {
        hi = hello.sayHi();
    }

    @Then("^it should answer with \"([^\"]*)\"$")
    public void it_should_answer_with(String expectedHi) {
        assertEquals(expectedHi, hi);
    }
}
EOF
```

Crie a classe `Hello`:

```
d=src/main/java/com/ladoservidor/cucumber/helloworld
mkdir -p $d
cat > $d/Hello.java <<'EOF'
package com.ladoservidor.cucumber.helloworld;

public class Hello {
    private final String greeting;

    public Hello(String greeting) {
        this.greeting = greeting;
    }
}
```

```
        public String sayHi() {
            return greeting + " World";
        }
    }
EOF
```

---

Reexecute os testes com o maven:

```
mvn test
```

---

Altere o arquivo `helloworld.feature` para utilizar o português:

```
d=src/test/resources/com/ladoservidor/cucumber/helloworld
cat > $d/helloworld.feature <<'EOF'
# language: pt
Funcionalidade: Diga Olá

    Cenário: Dizer "Olá Fulano!"
        Dado que eu tenho uma app que recebe "Paulo"
        Quando eu pedir que ela diga olá
        Então ela deveria responder "Olá Paulo!"
EOF
```

---

Altere o `RunCukesTest` para suportar o português:

```
patch -p1 <<'EOF'
--- ./src/test/java/com/ladoservidor/cucumber/helloworld/RunCukesTest.java
    2013-04-05 15:44:14.000000000 -0300
+++ ./HelloWorld.pt/src/test/java/com/ladoservidor/cucumber/helloworld/
RunCukesTest.java 2013-04-05 15:45:15.000000000 -0300
@@ -8,7 +8,8 @@ import org.junit.runner.RunWith;
    format = {
        "pretty",
        "html:target/cucumber-html-report",
-       "json-pretty:target/cucumber-json-report.json"
+       "json-pretty:target/cucumber-json-report.json",
+       "json:target/cucumber-pt.json"
    }
)
public class RunCukesTest {
EOF
```

---

Altere o `HelloStepdefs` para suportar o português:

```
patch -p1 <<'EOF'
--- ./src/test/java/com/ladoservidor/cucumber/helloworld/
HelloStepdefs.java 2013-04-05 15:44:14.000000000 -0300
+++ ./HelloWorld.pt/src/test/java/com/ladoservidor/cucumber/helloworld/
HelloStepdefs.java 2013-04-05 15:45:15.000000000 -0300
@@ -1,8 +1,8 @@
 package com.ladoservidor.cucumber.helloworld;

-import cucumber.api.java.en.Given;
-import cucumber.api.java.en.Then;
-import cucumber.api.java.en.When;
+import cucumber.api.java.pt.Dado;
+import cucumber.api.java.pt.Quando;
+import cucumber.api.java.pt.Entao;

 import static org.junit.Assert.assertEquals;

@@ -10,17 +10,17 @@ public class HelloStepdefs {
     private Hello hello;
     private String hi;

-    @Given("^I have a hello app with \"([^\"]*)\"$")
+    @Dado("^que eu tenho uma app que recebe \"([^\"]*)\"$")
     public void I_have_a_hello_app_with(String greeting) {
         hello = new Hello(greeting);
     }

-    @When("^I ask it to say hi$")
+    @Quando("^eu pedir que ela diga olá$")
     public void I_ask_it_to_say_hi() {
         hi = hello.sayHi();
     }

-    @Then("^it should answer with \"([^\"]*)\"$")
+    @Entao("^ela deveria responder \"([^\"]*)!\"$")
     public void it_should_answer_with(String expectedHi) {
         assertEquals(expectedHi, hi);
     }
 }
EOF
```

Altere o `Hello` para português:

```
patch -p1 <<'EOF'
```



```
--- ./src/main/java/com/ladoservidor/cucumber/helloworld/Hello.java
    2013-04-05 15:44:14.000000000 -0300
+++ ./HelloWorld.pt/src/main/java/com/ladoservidor/cucumber/helloworld/
Hello.java 2013-04-05 15:45:15.000000000 -0300
@@ -8,6 +8,6 @@ public class Hello {
    }

    public String sayHi() {
-        return greeting + " World";
+        return "Olá " + greeting;
    }
}
EOF
```

---

Reexecute os testes:

---

```
mvn test
```

---

---

# Chapter 6. Testes reais com o Arquillian

## 6.1. TDD com Arquillian: desenvolvendo uma aplicação Java EE do zero

- Referências:
  - [http://arquillian.org/guides/getting\\_started\\_pt/](http://arquillian.org/guides/getting_started_pt/)
  - [http://arquillian.org/guides/get\\_started\\_faster\\_with\\_forge\\_pt/](http://arquillian.org/guides/get_started_faster_with_forge_pt/)

## 6.2. Executando e testando a aplicação tasks-rs

- Referências:
  - <https://github.com/jboss-developer/jboss-eap-quickstarts/tree/6.3.x/tasks-rs>

### 6.2.1. Iniciando o JBoss e implantando a aplicação

Inicie o JBoss:

```
jboss_start
```

Vá para o diretório da aplicação tasks-rs e crie um usuário para a aplicação. Execute os comandos abaixo:

```
projetos && cd jboss-eap-6.3.0.GA-quickstarts/tasks-rs
add-user.sh -a -u 'quickstartUser' -p 'quickstartPwd1!' -g 'guest'
```

### 6.2.2. Testando a aplicação

Execute:

```
curl -i -u 'quickstartUser:quickstartPwd1!' -H "Content-Length: 0" -X POST
http://localhost:8080/jboss-tasks-rs/tasks/task1
```

Verifique a resposta esperada:

---

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Location: http://localhost:8080/jboss-tasks-rs/tasks/1
Content-Length: 0
Date: Thu, 05 Feb 2015 11:45:04 GMT
```

---

## 6.3. Executando e testando a aplicação kitchensink-angularjs

- Referências:
  - <http://www.jboss.org/quickstarts/wfk/kitchensink-angularjs/>

---

# Chapter 7. Test-Driven JavaScript Development

---

# Chapter 8. Referências

## 1. Artigos e apresentações (antigas) produzidos pelo autor:

- a. [Test-Driven Development \(TDD\) em Java \(Parte 1\)](#)<sup>1</sup> (*Ref. básica*)
- b. [TDD e BDD em Aplicações Java EE com JUnit, Arquillian, Selenium e Cucumber, parte 1](#)<sup>2</sup>
- c. [Tutorial: Testes reais de componentes Java EE com Arquillian e ShrinkWrap](#)<sup>3</sup>
- d. [3Plans.net - em ambiente Java EE 6 \(JBoss EAP, OpenShift\)](#)<sup>4</sup>
- e. [Processo de desenvolvimento guiado pelos testes](#)<sup>5</sup>

## 2. Artigos:

- a. [Test Driven Development](#)<sup>6</sup> (*Ref. básica*)
- b. [TDD | Caelum](#)<sup>7</sup> (*Ref. básica*)
- c. [Brief comparison of BDD frameworks](#)<sup>8</sup>

## 3. Livros:

- a. [Continuous Enterprise Development in Java - Testable Solutions with Arquillian](#)<sup>9</sup>
- b. [Test-driven Development: By Example](#)<sup>10</sup> (*Ref. básica*)
- c. [Pragmatic Unit Testing in Java 8 with JUnit](#)<sup>11</sup>
- d. [Enterprise Web Development: From Desktop to Mobile](#)<sup>12</sup>

<sup>1</sup> <https://dl.dropboxusercontent.com/u/345266/artigos/tdd/1/index.html>

<sup>2</sup> <http://blog.ladoservidor.com/2013/04/agilebrazil-1.html>

<sup>3</sup> <http://a.ladoservidor.com/tutoriais/arquillian-shrinkwrap/index.html>

<sup>4</sup> <http://a.paulojeronimo.info/3plans/docs/tutorial-javaee-6.html>

<sup>5</sup> <http://a.paulojeronimo.info/capes/processo-de-testes/index.html>

<sup>6</sup> [http://pt.wikipedia.org/wiki/Test\\_Driven\\_Development](http://pt.wikipedia.org/wiki/Test_Driven_Development)

<sup>7</sup> <http://tdd.caelum.com.br/>

<sup>8</sup> <http://java.dzone.com/articles/brief-comparison-bdd>

<sup>9</sup> <https://github.com/arquillian/continuous-enterprise-development>

<sup>10</sup> [http://books.google.com.br/books?id=gFgnde\\_vwMAC](http://books.google.com.br/books?id=gFgnde_vwMAC)

<sup>11</sup> <https://pragprog.com/book/utj2/pragmatic-unit-testing-in-java-8-with-junit>

<sup>12</sup> [http://enterprisewebbook.com/ch7\\_testdriven\\_js.html](http://enterprisewebbook.com/ch7_testdriven_js.html)

<sup>13</sup> <http://tddjs.com/>

- f. Test-Driven Development: Teste e Design no Mundo Real<sup>14</sup>

---

<sup>14</sup> <http://www.casadocodigo.com.br/products/livro-tdd>