Testes em aplicações Java EE com Arquillian

Paulo Jerônimo

Table of Contents

Chapter 1. Introdução

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

Este material utiliza scripts para que automatizam a montagem de um ambiente de desenvolvimento para aplicações Java EE[6] 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.

1 ./javaee-tutorial-testes.pdf
2 http://pt.wikipedia.org/wiki/Test_Driven_Development
3 http://pt.wikipedia.org/wiki/Behavior_Driven_Development
4 http://arquillian.org/guides/getting_started_pt/
5 https://angularjs.org/
6 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[1]). 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, acredita-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, julga-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[2] 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.

[1] http://fedoraproject.org
[2] 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 apresentada na execução 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 escrito da forma 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. Se, antes desse passo, você desejar manter o cache dos pacotes que serão baixados em tua máquina, execute:

```
sudo sed -i 's/^\(keepcache=\)0/\11/g' /etc/yum.conf
```

Atualize os pacotes:

```
sudo yum -y update
```

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 o 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 pacotes contendo utilitários que serão executados pelos scripts deste tutorial

Os scripts que você executará mais a frente necessitam da instalação de alguns pacotes de utilitários. Então, execute a seguinte instrução:

```
sudo yum -y install vim redhat-lsb-core patch
```

3.4. Instalando o Oracle JDK

Alguns sites, como os do Banco do Brasil[1] e do Itaú[2], dependem da instalação do Oracle JRE[3]. Então, é interessante ter esta JRE instalada. E, apesar do OpenJDK fazer parte da instalação padrão do Fedora 21, utilizaremos o Oracle JDK[4], neste material.

Para instalar o Oracle JRE (e JDK) utilizaremos o Fedy[5], executando os comandos a seguir:

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

[1] http://www.bb.com.br
[2] http://www.itau.com.br
[3] http://www.oracle.com/technetwork/java/javase/downloads/index.html
[4] http://www.oracle.com/technetwork/java/javase/downloads/index.html
[5] https://satya164.github.io/fedy/

```
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 --set java /usr/java/latest/jre/bin/java
sudo alternatives --set libjavaplugin.so.x86_64 /usr/java/latest/jre/lib/
amd64/libnpjp2.so
```

Pronto, agora testemos a execução de applets Java acessando a página "Verificar Versão do Java"[6] e também os sites dos bancos brasileiros.

3.5. Utilizando o Fedy para instalar softwares proprietários

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

> Essas instalações/configurações são opcionais no contexto deste tutorial.

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:

[6] https://www.java.com/pt_BR/download/installed.jsp

```
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
```

Durante execução do comando acima, em meu caso, ocorreram erros na tentativa de instalação de `dvd_playback`. Isso ocorreu pois não foi encontrado o pacote `libdvdcss`.

Chapter 4. Montando um ambiente para este tutorial

4.1. Baixando 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. Criando 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[1], siga os passos descritos em na página do produto[2]. 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
```

[1] http://tools.jboss.org/downloads/devstudio/index.html
[2] http://tools.jboss.org/downloads/devstudio/luna/8.0.2.GA.html

Chapter 5. Uma revisão, rápida e prática, sobre TDD e BDD

Talvez você queira dar uma olhada numa apresentação que fiz para a Capes, em 2013[1].

5.1. TDD sem JUnit, para os bravos

Leia o tutorial Test-Driven Development (TDD) em Java (Parte 1)[2] 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) {
```

[1] http://a.paulojeronimo.info/capes/processo-de-testes/index.html
[2] 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, criando a classe a seguir:

```
cat > AssertionFailedError.java <<EOF
public class AssertionFailedError extends Error {
    public AssertionFailedError(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 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 AssertionFailedError(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 AssertionFailedError(message);
     }

+    public static void assertEquals(String message, long expected, long
 actual) {
+        if (expected != actual) {
+            throw new AssertionFailedError(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 corrigí-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 AssertionFailedError(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[3] 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>
```

[3] 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 `HelloStepsdefs` 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/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)[1] (*Ref. básica*)
    b. TDD e BDD em Aplicações Java EE com JUnit, Arquillian, Selenium e Cucumber, parte 1[2]
    c. Tutorial: Testes reais de componentes Java EE com Arquillian e ShrinkWrap[3]
    d. 3Plans.net - em ambiente Java EE 6 (JBoss EAP, OpenShift)[4]
    e. Processo de desenvolvimento guiado pelos testes[5]
2. Artigos:
    a. Test Driven Development[6] (*Ref. básica*)
    b. TDD | Caelum[7] (*Ref. básica*)
    c. Brief comparison of BDD frameworks[8]
3. Livros:
    a. Continuous Enterprise Development in Java - Testable Solutions with Arquillian[9]
    b. Test-driven Development: By Example[10] (*Ref. básica*)
    c. Pragmatic Unit Testing in Java 8 with JUnit[11]
    d. Enterprise Web Development: From Desktop to Mobile[12]
    e. Test-Driven JavaScript Development[13]

[1] https://dl.dropboxusercontent.com/u/345266/artigos/tdd/1/index.html
[2] http://blog.ladoservidor.com/2013/04/agilebrazil-1.html
[3] http://a.ladoservidor.com/tutoriais/arquillian-shrinkwrap/index.html
[4] http://a.paulojeronimo.info/3plans/docs/tutorial-javaee-6.html
[5] http://a.paulojeronimo.info/capes/processo-de-testes/index.html
[6] http://pt.wikipedia.org/wiki/Test_Driven_Development
[7] http://tdd.caelum.com.br/
[8] http://java.dzone.com/articles/brief-comparison-bdd
[9] https://github.com/arquillian/continuous-enterprise-development
[10] http://books.google.com.br/books?id=gFgnde_vwMAC
[11] https://pragprog.com/book/utj2/pragmatic-unit-testing-in-java-8-with-junit
[12] http://enterprisewebbook.com/ch7_testdriven_js.html
[13] http://tddjs.com/

f. Test-Driven Development: Teste e Design no Mundo Real[14]

[14] http://www.casadocodigo.com.br/products/livro-tdd

Chapter 9. Extras

9.1. Contribuindo com atualizações e/ou correções neste material

Este material é gratuito e publicado, livremente, no site http://paulojeronimo.github.io/javaee-tutorial-testes. Esse site e a sua impressão em formato pdf[1] são gerados a partir de um código fonte escrito no formato Asciidoc[2] através da ferramenta Asciidoctor[3]. Um script bash (build[4]) é utilizado para esta geração.

São bem vindas as contribuições (atualizações e/ou correções) a este material, que é disponibilizado sob uma licença Creative Commons. Essas contribuições podem ser submetidas via pull request[5] no repositório do projeto.

Para instalar o Asciidoctor, execute:

```
gem install asciidoctor
```

Esse comando necessita de um ambiente Ruby instalado. Se você ainda não possui esse ambiente em teu Fedora, a melhor maneira de criá-lo é instalar o RVM[6] para, através dele, instalar o Ruby. Isso pode ser feito com os seguintes comandos:

```
gpg --keyserver hkp://keys.gnupg.net --recv-keys D39DC0E3
curl -sSL https://get.rvm.io | bash -s stable
source ~/.profile
rvm install 2.2.0
ruby --version
```

Estando no diretório que contém o seu clone do projeto (javaee-tutorial-testes), para gerar o arquivo index.html[7] e, em seguida, o arquivo javaee-tutorial-testes.pdf[8], execute os seguintes comandos:

1 ./javaee-tutorial-testes.pdf
2 http://en.wikipedia.org/wiki/AsciiDoc
3 http://asciidoctor.org
4 ./build
5 https://help.github.com/articles/using-pull-requests/
6 http://rvm.io/
7 ./index.html

```
./build
./build pdf
```

[8] ./javaee-tutorial-testes.pdf