CAPÍTULO 12

Apêndice Testes de interface com Selenium

"Além da magia negra, há apenas automação e mecanização"
— Federico Garcia Lorca

12.1 – ALTERANDO O TÍTULO DO GRÁFICO

Olhando para o nosso gráfico reparamos que ele possui como título o texto *Indicadores*, sendo que este não pode ser alterado. Seria interessante permitir que o usuário pudesse alterar esse título, colocando outro texto de sua preferência.

Podemos fazer isso adicionando mais um campo no formulário, sendo que este será um campo de texto, e para isso devemos utilizar a tag `<h:inputText>` do JSF, ou então a tag `<p:inputText>` do PrimeFaces:

```
<h:outputLabel value="Título Gráfico: "/>
<p:inputText id="titulo" value="#{argentumBean.titulo}" />
```

Também será necessário adicionar mais um atributo do tipo `String` à classe `ArgentumBean`, bem como seu getter e setter.

Outra mudança que precisamos fazer é passar o título digitado pelo usuário para o `GeradorModeloGrafico`. Podemos passar esta informação pelo construtor, ao instanciarmos o objeto `GeradorModeloGrafico`:

```
public void geraGrafico() {
  List<Candle> candles = new
CandleFactory().constroiCandles(this.negociacoes);
  SerieTemporal serie = new SerieTemporal(candles);

  GeradorModeloGrafico geradorGrafico = new GeradorModeloGrafico(
      serie, 2, serie.getUltimaPosicao(), titulo);
  geradorGrafico.plotaIndicador(defineIndicador());
```

```
  this.modeloGrafico = geradorGrafico.getModeloGrafico();
}
```

E no construtor da classe `GeradorModeloGrafico` recebemos o título e o guardamos em um novo atributo:

```
public class GeradorModeloGrafico {

  private SerieTemporal serie;
  private int comeco;
  private int fim;
  private LineChartModel modeloGrafico;
  private String tituloGrafico;

  public GeradorModeloGrafico(SerieTemporal serie, int comeco, int fim,
      String tituloGrafico) {
    this.serie = serie;
    this.comeco = comeco;
    this.fim = fim;
    this.tituloGrafico = tituloGrafico;
    this.modeloGrafico = new LineChartModel();
  }

  //restante do código
```

E por fim no método `plotaIndicador` devemos atribuir o título do gráfico ao objeto `LineChartModel`:

```
public void plotaIndicador(Indicador indicador) {
  LineChartSeries chartSerie = new LineChartSeries(indicador.toString());

  for (int i = this.comeco; i <= this.fim; i++) {
    double valor = indicador.calcula(i, this.serie);
    chartSerie.set(Integer.valueOf(i), Double.valueOf(valor));
  }
  this.modeloGrafico.addSeries(chartSerie);
  this.modeloGrafico.setLegendPosition("w");
  this.modeloGrafico.setTitle(tituloGrafico);
}
```

12.2 – Validação com JSF

Podemos agora definir um título para cada geração de gráfico, mas temos um problema: nada impede que o usuário deixe o título em branco, o que não faz sentido para nossa aplicação.

Preenchimento obrigatório através do atributo required

O JSF vem nos socorrer. Se quisermos tornar o preenchimento do título obrigatório, basta adicionarmos o atributo `required="true"` no componente

`p:inputText`. A mesma coisa é válida para o `h:inputText` da especificação.

```
<p:inputText id="titulo" value="#{argentumBean.titulo}" required="true"/>
```

Quando o formulário for submetido com o título em branco, o JSF automaticamente gerará uma mensagem avisando ao usuário que o campo deve ser preenchido. O problema é que ainda não definimos qual componente será utilizado para mostrar essa mensagem.

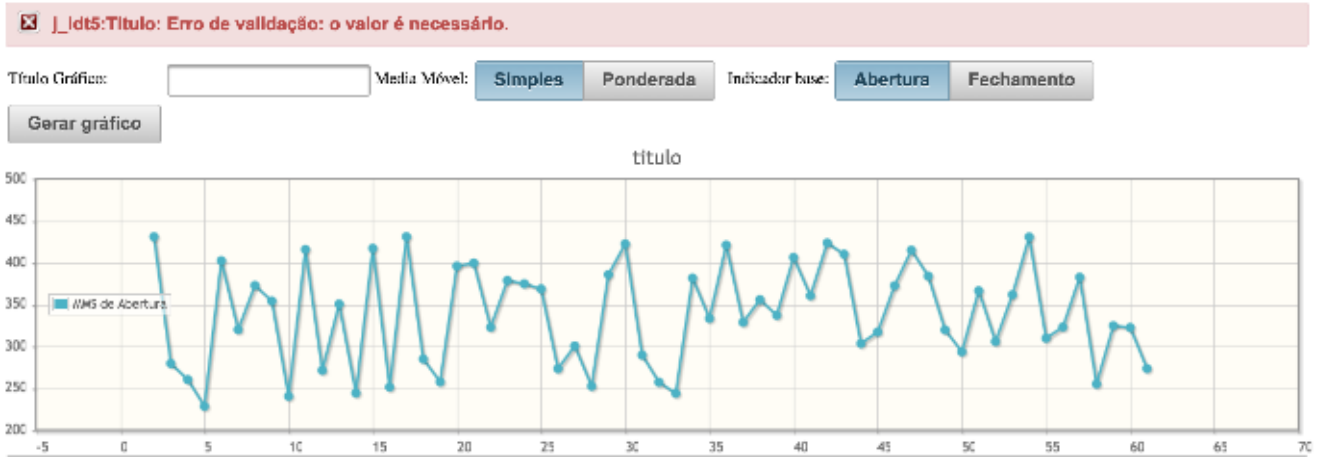
Exibindo mensagens com p:messages

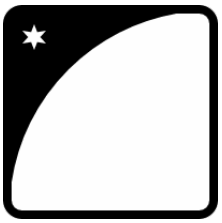
O PrimeFaces possui o componente `p:messages` que exibe para o usuário as mensagens geradas pelo JSF, em nosso caso, as de validação. O objetivo aqui é fazer com que a mensagem de preenchimento obrigatório, inclusive outras mensagens que possam ser geradas, seja exibida para o usuário:

```
....
<h:form>
  <p:messages autoUpdate="true" />
....
```

O atibuto `autoUpdate="true"` indica ao PrimeFaces que este componente deve ser atualizado automaticamente em caso de requisições `AJAX`.

Pronto, quando o formulário for submetido com o campo título em branco o JSF exibirá automaticamente a mensagem de validação:





A Caelum oferece o **curso FJ-22** presencial nas cidades de São Paulo, Rio de Janeiro e Brasília, além de turmas incompany.

Consulte as vantagens do curso *Lab. Java com Testes, JSF e Design Patterns*.

12.3 - Introdução aos testes de aceitação

Nos capítulos anteriores fizemos uso dos testes de unidade, será que eles são suficientes para garantir que a aplicação funcione da maneira esperada? Em um projeto web, uma das partes mais importantes para o funcionamento desejado é a parte de apresentação, já que é a partir dela que toda a comunicação do cliente com o nosso sistema será feita.

Verificar se as funcionalidades de um sistema estão se comportando corretamente sem que tenhamos de testar manualmente a aplicação, abrindo um navegador, navegando por ele e visualizando os resultados são tarefas que são realizadas pelos *Acceptance Testing* (ou Testes de Aceitação).

Temos que testar o nosso código, mas em uma aplicação web fica difícil testar a camada de apresentação, o resultado final. Como testar a compatibilidade entre browsers diferentes de maneira automatizada? Em geral, como testar se a página renderizada tem o resultado desejado?

Para isso ser possível, existem algumas ferramentas que permitem tal trabalho, simulando a interação de um usuário com a aplicação. Esse é o caso do **Selenium** que, com a assistência do **JUnit**, pode facilitar o trabalho de escrever tais testes.

12.4 - Como funciona?

Para demonstrar como o Selenium funciona, testaremos se a validação do campo *titulo* funciona corretamente, isto é, o formulário não poderá ser submetido caso ele esteja em branco.

A primeira coisa que uma pessoa faz para testar a aplicação é abrir um browser. Com Selenium, precisamos apenas da linha abaixo:

```
WebDriver driver = new FirefoxDriver();
```

Nesse caso, abrimos o Firefox. A ideia aqui é igual ao mundo JDBC onde o Driver abstrai detalhes sobre o funcionamento do banco de dados. Em nosso caso o driver se preocupa com a comunicação com o navegador.

Há outros drivers disponíveis para Chrome, Internet Explorer, Safari e até para Android ou iPhone. O que precisa mudar é apenas a implementação concreta da interface `WebDriver`, por exemplo:

```
WebDriver driver = new InternetExplorerDriver();
```

HtmlUnitDriver

Há uma outra implementação do `WebDriver` disponível que nem precisa abrir um navegador. Ela se chama `HtmlUnitDriver` e simula o navegador em memória. Isso é muito mais rápido mas não testa a compatibilidade do navegador fidedignamente:

```
WebDriver driver = new HtmlUnitDriver();
```

Essa alternativa é bastante utilizada em ambientes de integração que fazem uso de um cloud, já que não temos disponíveis um browser para teste.

O próximo passo é indicar qual página queremos abrir através do navegador. O driver possui um método `navigate().to()` que recebe a URL:

```
WebDriver driver = new FirefoxDriver();

driver.navigate().to("http://localhost:8080/fj22-argentum-web/index.xhtml");
```

Com a página aberta, faremos a inserção de um valor em branco no campo `titulo`. Para isso, precisamos da referência deste elemento em nosso código, o que é feito pelo método `findElement`.

Este método recebe como parâmetro um critério de busca. A classe `By` possui uma série de métodos estáticos. Pesquisaremos pelo `id`:

```
WebDriver driver = new FirefoxDriver();

driver.navigate().to("http://localhost:8080/fj22-argentum-web/index.xhtml");

WebElement titulo = driver.findElement(By.id("titulo"));
```

Agora que temos o elemento, podemos preenchê-lo com o texto que quisermos

através do método *sendKeys*. Em nosso caso, deixaremos o campo em branco passando uma string vazia:

```
WebDriver driver = new FirefoxDriver();

driver.navigate().to("http://localhost:8080/fj22-argentum-web/index.xhtml");

WebElement titulo = driver.findElement(By.id("titulo"));

titulo.sendKeys("");
```

Quando o formulário for submetido e o campo `titulo` estiver em branco, esperamos que o sistema mostre a mensagem *Erro de Validação*.

Antes de testarmos se tudo esta correto, precisamos submeter o formulário. O objeto `WebElement` possui o método `submit` que faz exatamente isso:

```
WebDriver driver = new FirefoxDriver();

driver.navigate().to("http://localhost:8080/fj22-argentum-web/index.xhtml");

WebElement titulo = driver.findElement(By.id("titulo"));

titulo.sendKeys("");
titulo.submit();
```

Para termos certeza de que o sistema esta validando, pediremos ao Selenium que procure pelo texto "Erro de Validação". Primeiro, precisamos evocar o método `getPageSource`, que nos permite procurar por algo no código da página, e logo em seguida o método `contains`, que retorna *true* ou *false*:

```
WebDriver driver = new FirefoxDriver();

driver.navigate().to("http://localhost:8080/fj22-argentum-web/index.xhtml");

WebElement titulo = driver.findElement(By.id("titulo"));

titulo.sendKeys("");
titulo.submit();

boolean existeMensagem = driver.getPageSource().contains("Erro de
validação");
```

Por fim, após o formulário ter sido submetido, precisamos saber se tudo saiu conforme o planejado, o que pode ser feito através do método estático `assertTrue` da classe `Assert`, que recebe como parâmetro um boolean que indica a presença ou não do texto procurado pelo método `contains`:

```
WebDriver driver = new FirefoxDriver();
```

```
driver.navigate().to("http://localhost:8080/fj22-argentum-web/index.xhtml");

WebElement titulo = driver.findElement(By.id("titulo"));

titulo.sendKeys("");
titulo.submit();

boolean existeMensagem = driver.getPageSource().contains("Erro de
validação");

Assert.assertTrue(existeMensagem);
```

Pronto, podemos verificar no próprio Eclipse se o teste passou se ele estiver verde. Se algo der errado, como de costume, a cor vermelha será utilizada.

Podemos usar agora o método `driver.close()` para fechar a janela do navegador.

```
WebDriver driver = new FirefoxDriver();

driver.navigate().to("http://localhost:8080/fj22-argentum-web/index.xhtml");

WebElement titulo = driver.findElement(By.id("titulo"));

titulo.sendKeys("");
titulo.submit();

boolean existeMensagem = driver.getPageSource().contains("Erro de validação);

Assert.assertTrue(existeMensagem);

driver.close();
```

12.5 – Trabalhando com diversos testes de aceitação

É comum termos dois ou mais testes de aceitação em nossa bateria de testes. Teríamos então que abrir uma janela do navegador em cada método, e, após os testes, fechá-la. Com isso, estaríamos repetindo muito código! O **Selenium** nos permite fazer isso de uma forma mais fácil. Primeiro, criaremos o método `setUp`, e o anotaremos com `@Before`.

```
@Before
public void setUp() {
  driver = new FirefoxDriver();
}
```

Este método será invocado antes de cada teste, sempre abrindo uma nova janela. Analogamente, existe a anotação `@After`, que indica que o método será invocado após cada teste. Agora, precisamos fechar essas janelas:

```
@After
public void tearDown() {
  driver.close();
}
```


12.6 - Para saber mais: Configurando o Selenium em casa

Caso você esteja fazendo esse passo de casa, é preciso baixar algumas JARs para o funcionamento dessa aplicação, usaremos as seguintes versões:

- commons-exec-1.1.jar
- commons-logging-1.1.3.jar
- gson-2.3.jar
- guava-18.0.jar
- httpclient-4.3.4.jar
- httpcore-4.3.2.jar
- selenium-java-2.44.0.jar

Para mais informações, você pode consultar o site http://docs.seleniumhq.org/

12.7 - Exercícios: Testes de aceitação com Selenium

Vamos criar nosso primeiro teste com Selenium:

1. Primeiramente devemos alterar nossa página adicionando o campo para o título. Adicione o campo de texto logo após a abertura da tag `<h:panelGrid>` no

arquivo `index.xhtml`:

```
<h:panelGrid columns="5">
  <h:outputLabel value="Título Gráfico: "/>
  <p:inputText id="titulo" value="#{argentumBean.titulo}" required="true"/>
```

2. Como adicionamos mais um campo no formulário, altere a quantidade de colunas para 6 no componente `<h:panelGrid>`:

```
<h:panelGrid columns="6">
```

3. Para facilitar a localização do campo título nos testes, vamos atribuir um `id` ao componente `<h:form>`

```
<h:form id="dadosGrafico">
  <h:panelGrid columns="6">
```

4. Devemos agora alterar a classe `ArgentumBean` adicionando o atributo `titulo`, juntamente como seu getter e setter:

```
@ManagedBean
@ViewScoped
public class ArgentumBean implements Serializable {

  private String titulo;

  public String getTitulo() {
    return titulo;
  }
  public void setTitulo(String titulo) {
    this.titulo = titulo;
  }

  //restante do código
```

5. Ainda na classe `ArgentumBean` vamos alterar o método `geraGrafico`, passando o título como parâmetro para o construtor do `GeradorModeloGrafico`:

```
public void geraGrafico() {
  List<Candle> candles = new
CandleFactory().constroiCandles(this.negociacoes);
  SerieTemporal serie = new SerieTemporal(candles);

  GeradorModeloGrafico geradorGrafico = new GeradorModeloGrafico(serie, 2,
    serie.getUltimaPosicao(), titulo);
  geradorGrafico.plotaIndicador(defineIndicador());
  this.modeloGrafico = geradorGrafico.getModeloGrafico();
}

//restante do código
```

6. Agora na classe `GeradorModeloGrafico` vamos alterar o construtor para receber o título como parâmetro, e atribuí-lo a um novo atributo:

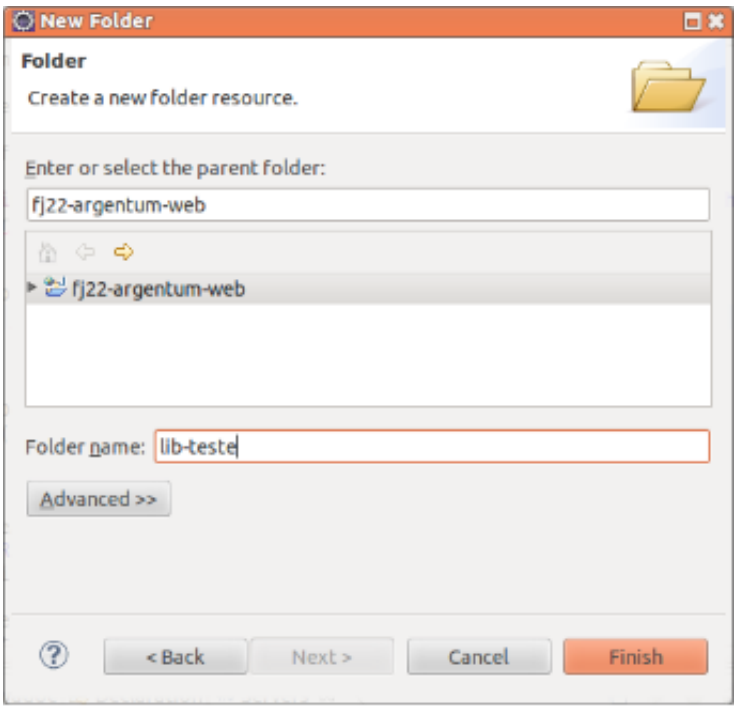
```
public class GeradorModeloGrafico {

private SerieTemporal serie;
private int comeco;
private int fim;
private LineChartModel modeloGrafico;
private String tituloGrafico;

public GeradorModeloGrafico(SerieTemporal serie, int comeco, int fim,
    String tituloGrafico) {
  this.serie = serie;
  this.comeco = comeco;
  this.fim = fim;
  this.tituloGrafico = tituloGrafico;
  this.modeloGrafico = new LineChartModel();
}
```

7. E no método `plotaIndicador` passamos o título como parâmetro no método `setTitle` do objeto `modeloGrafico`:

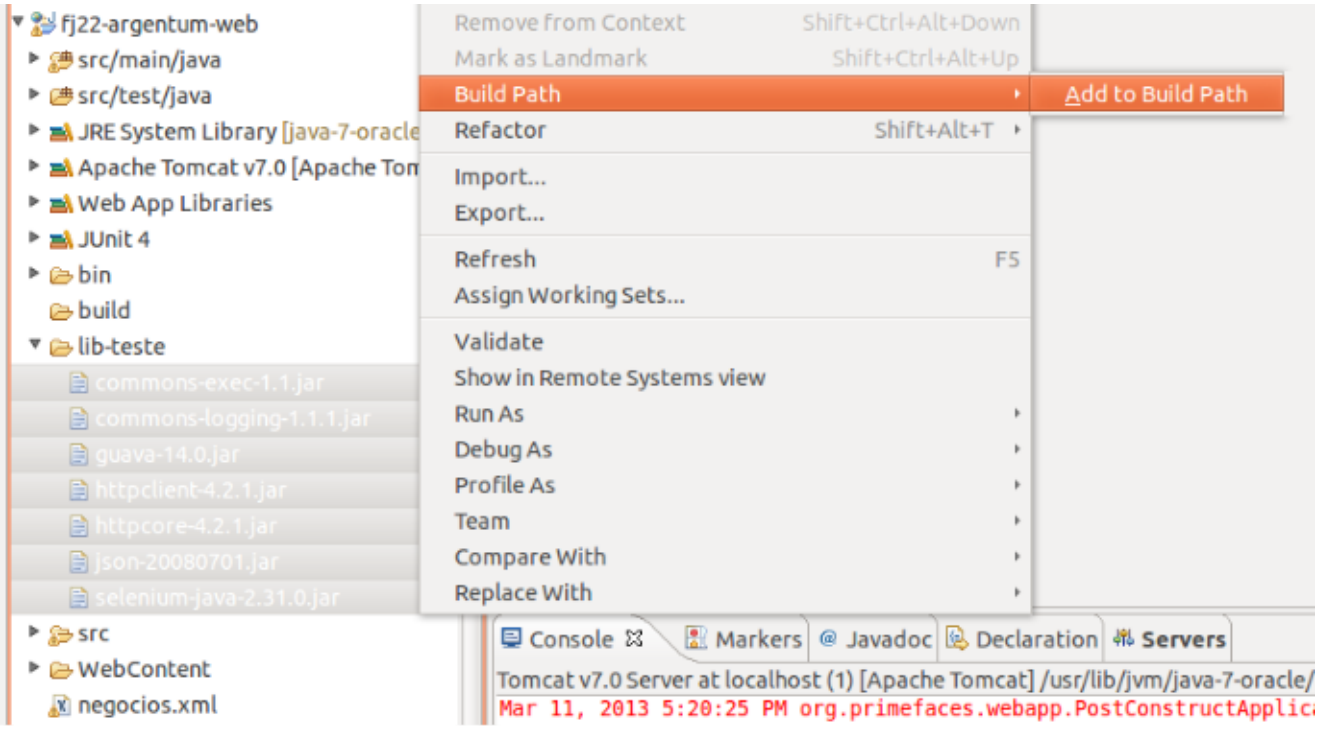
```
public void plotaIndicador(Indicador indicador) {
  LineChartSeries chartSerie = new LineChartSeries(indicador.toString());

  for (int i = this.comeco; i <= this.fim; i++) {
    double valor = indicador.calcula(i, this.serie);
    chartSerie.set(Integer.valueOf(i), Double.valueOf(valor));
  }
  this.modeloGrafico.addSeries(chartSerie);
  this.modeloGrafico.setLegendPosition("w");
  this.modeloGrafico.setTitle(tituloGrafico);
}
```

8. Agora vamos colocar os jars do Selenium no nosso projeto, eles se encontram na pasta `Desktop/caelum/22/selenium-jars/`:

Para adicioná-los ao projeto, crie uma nova pasta chamada `lib-teste` na raiz do projeto e copie os jars para dentro dela.



Logo em seguida adicione-os ao *Build Path*:



9. Crie a classe usando **ctrl + N** `Class`, chamada `GeraGraficoTest` no *source folder* *src/test/java* e dentro do pacote `br.com.caelum.argentum.aceitacao`:

10. A classe terá dois atributos. O primeiro com a URL da página que queremos testar e o segundo o WebDriver, o objeto que nos permite manipular o navegador.

```
public class GeraGraficoTest {

  private static final String URL =
    "http://localhost:8080/fj22-argentum-web/index.xhtml";

  private WebDriver driver;
```

```
}
```

11. Vamos criar o método `testeAoGerarGraficoSemTituloUmaMensagemEhApresentada` e anotá-lo com `@Test` para indicar que ele deve ser chamado quando o teste for executado:

```
public class GeraGraficoTest {

  private static final String URL =
    "http://localhost:8080/fj22-argentum-web/index.xhtml";

  private WebDriver driver;

  @Test
  public void testeAoGerarGraficoSemTituloUmaMensagemEhApresentada() {
  }
}
```

12. Usaremos o WebDriver para abrir uma nova janela do Firefox e acessar a URL do projeto, e usaremos o método `driver.close()` para fechar a janela

```
public class GeraGraficoTest {

  private static final String URL =
    "http://localhost:8080/fj22-argentum-web/index.xhtml";

  private WebDriver driver;

  @Test
  public void testeAoGerarGraficoSemTituloUmaMensagemEhApresentada() {
    driver = new FirefoxDriver();
    driver.navigate().to(URL);

    driver.close();
  }
}
```

13. Estamos testando se a mensagem de erro aparece quando submetemos um formulário com o título. Para isso, primeiro precisamos capturar o elemento do título em um `WebElement`. Como estamos trabalhando com **JSF**, devemos lembrar que ele concatena o *id* do formulário com o id dos inputs. Por conseguinte, devemos procurar o elemento pelo id `dadosGrafico:titulo`

```
public class GeraGraficoTest {

  private static final String URL =
    "http://localhost:8080/fj22-argentum-web/index.xhtml";

  private WebDriver driver;

  @Test
  public void testeAoGerarGraficoSemTituloUmaMensagemEhApresentada() {
    driver = new FirefoxDriver();
```

```
    driver.navigate().to(URL);
    WebElement titulo = driver.findElement(By.id("dadosGrafico:titulo"));

    titulo.sendKeys("");
    titulo.submit();

    boolean existeMensagem = driver.getPageSource().contains(
      "Erro de validação");

    Assert.assertTrue(existeMensagem);

    driver.close();
  }
}
```

14. Vemos que usamos `driver = new FirefoxDriver()` para abrir uma janela (um WebDriver) do navegador, e `driver.close()` para fechar a janela. Caso formos escrever mais testes, precisaremos abrir e fechar o navegador novamente. Para podermos reaproveitar esse código, podemos colocá-los em blocos separados e usar as anotações `@Before`, para executá-lo antes de cada método, e `@After`, para executá-lo após cada método.

```
public class GeraGraficoTest {

  private static final String URL =
    "http://localhost:8080/fj22-argentum-web/index.xhtml";

  private WebDriver driver;

  @Before
  public void setUp() {
    driver = new FirefoxDriver();
  }

  @After
  public void tearDown() {
    driver.close();
  }

  @Test
  public void testeAoGerarGraficoSemTituloUmaMensagemEhApresentada() {
    driver.navigate().to(URL);
    WebElement titulo = driver.findElement(By.id("dadosGrafico:titulo"));

    titulo.sendKeys("");
    titulo.submit();

    boolean existeMensagem = driver.getPageSource().contains(
      "Erro de validação");

    Assert.assertTrue(existeMensagem);
  }
}
```

CAPÍTULO ANTERIOR:

A API de Reflection

Você encontra a Caelum também em:

Blog Caelum

Cursos Online

Facebook

Newsletter

Casa do Código

Twitter