

Desenvolvimento Web II

Aula 03 - Validação de Campos no JSF

Apresentação

Olá! Seja bem-vindo(a) a nossa terceira aula sobre o framework JSF. Já que conhecemos as principais TAGS do JSF e sabemos como utilizá-las, iremos aprender como realizar validações de campos com esse framework. Essas validações são importantes, pois refletem as regras de negócio das nossas aplicações. Ao final da aula, praticaremos desenvolvendo uma aplicação JSF com validações de campos. Então, vamos começar?

Objetivos

- Entender a motivação para realizar validações em sistemas de informações;
- Aprender as diferentes formas de realizar validações com o framework JSF.

Validação

As validações das entradas de dados nos sistemas de informações são muito comuns, e têm o objetivo de verificar se todas as informações necessárias estão presentes para que o usuário possa continuar com a execução de uma determinada funcionalidade. Por exemplo, é comum os sites que tenham funcionalidade de cadastro de usuários validarem se o usuário informou seu login, seu e-mail, etc. Isso porque essas informações são consideradas necessárias para a realização do cadastro de usuário.

Além de validar se um determinado dado necessário para a nossa funcionalidade em nossos sistemas de informação web está presente, precisamos também retornar para o usuário uma mensagem amigável informando-o do erro. Imagine que você está utilizando um sistema de informação web, tentando fazer o seu cadastro e ao clicar no botão confirmar o mesmo não avança nem te informa porque não está avançando. Mesmo que esse comportamento ocorra devido a uma informação não inserida por você, a falta da mensagem de validação complica a realização do seu cadastro. Dessa forma, a validação das informações inseridas e a mensagem das mesmas tem o objetivo de guiar o usuário para realização com sucesso de uma determinada funcionalidade.

Portanto, validar informações envolve duas tarefas principais:

1. Verificar se as informações necessárias para continuação da funcionalidade estão presentes, ou seja, foram informadas pelo usuário, e estão no formato correto;
2. Exibir mensagens de validação quando alguma informação estiver faltando (não informada pelo usuário) ou no formato incorreto. Ao exibir essas mensagens, é importante mantermos os dados já informados pelo usuário que estão corretos.

Interessante, não é? Essas tarefas de validação são muito importantes para guiarmos a execução de uma funcionalidade. Mas você pode estar se perguntando: “Mas, professor, como eu sei que dados no meu sistema de informação web eu devo validar?” Ótima pergunta! O que determina que dados você deve validar em um

sistema de informação web é o negócio envolvido no problema que seu sistema vai resolver. Geralmente, o cliente determina quais funcionalidades deverão ser validadas baseado no problema que ele quer que o sistema de informação resolva.

Então, o JSF possui duas formas de validação, as quais veremos mais detalhadamente a seguir:

- Declarativa: utilizando os validadores padrões do JSF;
- Imperativa: método de validação no managed bean e classes validadoras que implementam a interface `javax.faces.validator.Validator`.

Validação Declarativa Usando Tags JSF

Como vimos na aula anterior, o JSF disponibiliza um conjunto de TAGs para que possamos adicionar componentes nas nossas páginas JSF (XHTML). Essas TAGs também permitem que façamos validações nos dados inseridos nas páginas. Dependendo do tipo do campo associado ao componente JSF, poderemos realizar validações de campo obrigatório, tamanho do dado do campo, intervalo de valor e validações, usando expressões regulares. Veremos agora exemplos de cada um desses tipos.

Validação de campo obrigatório

A validação de campo obrigatório verifica se um determinado campo não está vazio durante a submissão do formulário. A Listagem 1 apresenta um exemplo de validação de campo obrigatório no componente JSF.

```
1 <h:inputText id="nome" value="#{pessoaMBean.pessoa.nome}" title="Nome" required="true"
2   requiredMessage="Nome: campo obrigatório!"/>
```

Listagem 1 - Exemplo de validação de campo obrigatório na Tag `h:inputText`.

Validação de intervalo de campos numéricos

A validação de intervalo de campos numéricos verifica se um campo do tipo `long` ou `double` está entre um determinado intervalo de valores. Para realizar esse tipo de validação, é necessário que utilizemos as tags **`validateDoubleRange`** e **`validateLongRange`**, presentes no conjunto Tags JSF Core. Poderíamos utilizar essas validações, por exemplo, se quiséssemos validar um campo de nota no qual o aluno só pudesse ter notas entre 0 e 10. As listagens 2 e 3 apresentam exemplos de utilização dessas tags de validação.

```
1 <h:inputText value="#{alunoMBean.aluno.nota}" size="3" required="true"
2   requiredMessage="Nota: Campo obrigatório."
3   validatorMessage="Nota entre 0.0 e 10.0.">
4   <f:validateDoubleRange minimum="0.0" maximum="10.0"/>
5 </h:inputText>
```

Listagem 2 - Exemplo de validação do campo de intervalo de valores com a tag `validateDoubleRange`.

```
1 <h:inputText id="idade" value="#{pessoaMBean.idade}"
2   validatorMessage="A idade deve ser entre 26 e 45 anos.">
3   <f:validateLongRange minimum="26" maximum="45" />
4 </h:inputText>
```

Listagem 3 - Exemplo de validação do campo de intervalo de valores com a tag `validateLongRange`.

Observe que em ambas as Tags utilizamos os atributos **`minimum`** e **`maximum`** para determinar o intervalo no qual se deseja a validação. Para definição da mensagem de validação, utilizamos o atributo **`validatorMessage`**.

Validação de tamanho de campos de texto

Verifica se um campo de texto associado a um componente JSF tem um determinado tamanho. Para realizar esse tipo de validação, é necessário que utilizemos a tag **`validateLength`**, presente no conjunto Tags JSF Core. A Listagem 4

apresenta um exemplo de utilização dessa tag. Observe os atributos **minimum** e **maximum** para determinar o tamanho mínimo e máximo para o campo de texto que desejamos validar.

```
1 <h:inputText id="login" value="#{usuario.login}"
2   validatorMessage="Login deve ter entre 5 e 10 caracteres.">
3   <f:validateLength minimum="5" maximum="10" />
4 </h:inputText>
```

Listagem 4 - Exemplo de validação do tamanho de campo de texto com a tag `validateLength`.

Validação Imperativa

As validações declarativas são simples de serem utilizadas, mas geralmente não são suficientes para as diferentes regras de negócio em um sistema de informação. Muitas vezes, não queremos validar apenas a não existência de uma informação em um determinado campo. Existem casos que precisamos ir além disso, por exemplo: verificar se uma determinada matrícula já existe em um banco de dados durante um cadastro de aluno.

Esses casos estão associados às regras de negócio da aplicação e para eles, onde a validação declarativa não é suficiente, recorreremos à validação imperativa. Para esse tipo de validação podemos criar métodos nos controllers JSF (ManagedBeans) ou criar validadores customizados.

Validação usando métodos nos controllers

Para a realização de validação nos controllers, temos duas formas de fazer. Uma delas é colocar a regra de validação dentro do método associado à ação de um componente com um botão ou link. Essa forma é fácil de fazer e não exige que criemos nenhum outro método. As listagens 5 e 6 apresentam respectivamente um trecho de código referente a um botão e um método com sua validação.

```
1 <h:commandButton value="Cadastrar" action="#{cadastroMBean.cadastrar}" />
```

Listagem 5 - Exemplo de um botão associado ao método `cadastrar` do controller `CadastroMBean`.

```

1 public String cadastrar() {
2     //validando se a pessoa já existe no banco de dados
3     Pessoa pessoaBanco = dao.findPessoa(pessoa.getCpf());
4     if(pessoaBanco != null) {
5         FacesMessage msg = new FacesMessage("Essa pessoa já existe no banco de dados.");
6         msg.setSeverity(FacesMessage.SEVERITY_ERROR);
7         FacesContext.getCurrentInstance().addMessage("", msg);
8         return "/cadastro.jsf";
9     }
10    return "/continue.jsf";
11 }

```

Listagem 6 - Conteúdo do método cadastrar do controller CadastroMBean com validação.

Observe na Listagem 6, que para incluirmos as mensagens de validação, precisamos instanciar um objeto `FacesMessage`. Esse objeto encapsula o texto da mensagem e o tipo da mesma (erro, informação e aviso). Sempre que fizermos validações em métodos de ação será necessário instanciar objetos `FacesMessage`.

A outra forma de validar funcionalidades é criar métodos que possuam a assinatura definida na Listagem 7. Criar um método de validação dessa forma possibilitará utilizar o atributo **validator**, presentes nos componentes HTML do JSF, para associar uma regra de negócio a um campo do formulário. Vejamos como ficaria a mesma validação proposta no exemplo das listagens 5 e 6 se utilizássemos métodos associados aos campos do formulário. A Listagem 8 apresenta como ficaria o método para validação caso a pessoa já exista no banco de dados.

```

1 public void meuMetodo(FacesContext context, UIComponent component, java.lang.Object value){ ..

```

Listagem 7 - Assinatura dos métodos para validação associada a componentes.

```

1 public void validarPessoa(FacesContext context, UIComponent component, Object value) throws Va
2     String cpf = (String) value;
3     Pessoa pessoaBanco = dao.findPessoa(pessoa.getCpf());
4     if(pessoaBanco != null) {
5         String msg = "Essa pessoa já existe no banco de dados.";
6         throw new ValidatorException
7             (new FacesMessage(FacesMessage.SEVERITY_ERROR, msg, msg));
8     }
9 }

```

Listagem 8 - Método para validação associada a um componente.

Observe na Listagem 8 que o método dispara uma exceção de validação (ValidatorException) caso já exista uma pessoa cadastrada no banco de dados com o mesmo CPF. Para que essa validação funcione, precisamos associar esse método de validação (validarPessoa) com o campo do formulário que receberá o CPF informado pelo usuário. A Listagem 9 apresenta essa associação realizada através do atributo **validator**.

```
1 <h:inputText value="#{cadastroMBean.pessoa.cpf}"
2   size="11" validator="#{cadastroMBean.validarPessoa}" />
```

Listagem 9 - Campo de texto associado ao método de validação definido na Listagem 8.

Com isso, ao submeter o formulário esse campo de CPF será validado pelo método validarPessoa.

E, então, você gostou das formas de validação com métodos? Qual das duas achou melhor? Por quê?

Validação usando componentes customizados

Seguimos agora para outra forma de validação imperativa a qual permite que criemos validadores customizados para nossas aplicações. A criação de validadores customizados é útil quando temos uma determinada validação para um campo que se repetirá em mais de um formulário. Por exemplo, suponha que você tenha um campo para e-mail na sua aplicação o qual é solicitado ao usuário em diversas páginas diferentes. Nesse cenário, criar um validador customizado permitirá a centralização das verificações dos e-mails válidos. Para criar um validador customizado, precisamos:

1. Criar uma classe Java e anotá-la com @FacesValidator. A inclusão dessa anotação registra a classe como um validador JSF;
2. Além de anotar a classe, é necessário que ela implemente a interface javax.faces.validator.Validator. Ao implementar essa interface, precisamos definir o método validate, que possui a assinatura: public void validate(FacesContext context, UIComponent component, Object value) throws ValidatorException;
3. Na página JSF (xhtml), no componente a ser validado, precisamos incluir a tag validator, especificando o identificador do validador (validatorId).

Entendeu? As listagens 10 e 11, mostram respectivamente um simples validador de e-mail e sua utilização em um trecho de uma página JSF (xhtml).

```
1 import javax.faces.application.FacesMessage;
2 import javax.faces.component.UIComponent;
3 import javax.faces.context.FacesContext;
4 import javax.faces.validator.FacesValidator;
5 import javax.faces.validator.Validator;
6 import javax.faces.validator.ValidatorException;
7
8 @FacesValidator("emailValidator")
9 public class EmailValidator implements Validator{
10
11     @Override
12     public void validate(FacesContext context, UIComponent component, Object value) throws ValidationException {
13         String email = value.toString();
14         if(!email.contains("@")) {
15             FacesMessage msg = new FacesMessage(" E-mail inválido.",
16             "Formato: abcd@abc.com");
17             msg.setSeverity(FacesMessage.SEVERITY_ERROR);
18             throw new ValidatorException(msg);
19         }
20     }
21 }
22 }
```

Listagem 10 - Validador de e-mail.

```
1 <h:inputText value="#{cadastroMBean.pessoa.email}" size="20">
2     <f:validator validatorId="emailValidator" />
3 </h:inputText>
```

Listagem 11 - Utilização do validador de e-mail em um campo JSF.

Criando Uma Terceira Aplicação Web

Agora que já sabemos como realizar validações nas nossas aplicações JSF, vamos criar nossa terceira aplicação web. Essa aplicação será um formulário para cadastro de clientes onde realizaremos os vários tipos de validações. Então, vamos lá! Crie um projeto dinâmico web chamado AulaValidacoesJSF. Você poderá seguir os mesmos passos definidos na aula 1 e 2 para criação de um projeto JSF, sendo necessário definir o nome do projeto e o servidor de aplicação de destino.

Criado o projeto, coloque dentro da pasta lib o jar do JSF (javax.faces-2.2.8.jar) e em seguida crie o arquivo web.xml, que deverá ser adicionado na pasta WEB-INF. Esse arquivo conterá a definição da servlet do JSF, chamada de Faces Servlet, e o seu mapeamento. A Listagem 12 apresenta o conteúdo do web.xml. Veja que são os mesmos procedimentos utilizados nas aulas 1 e 2 para criarmos nossos projetos.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3 xmlns="http://java.sun.com/xml/ns/javaee"
4 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5 http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
6
7   <display-name>AulaValidacoesJSF</display-name>
8   <welcome-file-list>
9     <welcome-file>index.jsf</welcome-file>
10  </welcome-file-list>
11
12  <servlet>
13    <servlet-name>Faces Servlet</servlet-name>
14    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
15    <load-on-startup>1</load-on-startup>
16  </servlet>
17
18  <servlet-mapping>
19    <servlet-name>Faces Servlet</servlet-name>
20    <url-pattern>*.jsf</url-pattern>
21  </servlet-mapping>
22
23 </web-app>
```

Listagem 12 - Código do web.xml da aplicação AulaValidacoesJSF.

Conseguiu? Ótimo! Agora vamos criar nossos pacotes:

- br.ufrn.imd.dominio: no qual iremos adicionar à entidade Cliente;
- br.ufrn.imd.controllers: no qual iremos adicionar o controller ClienteMBean;
- br.ufrn.imd.validadores: no qual iremos adicionar o validador customizado ValidadorCPF.

A classe Cliente terá os atributos: nome, endereço, data de nascimento, sexo, CPF, telefone e celular. A Listagem 13 apresenta o código dessa classe que deverá estar no pacote **br.ufrn.imd.dominio**.

```
1 package br.ufrn.imd.dominio;
2 import java.util.Date;
3
4 public class Cliente {
5
6     private String nome;
7     private String endereco;
8     private Date dataNascimento;
9     private String sexo;
10    private String cpf;
11    private String telefone;
12    private String celular;
13
14    public String getNome() {
15        return nome;
16    }
17
18    public void setNome(String nome) {
19        this.nome = nome;
20    }
21
22    public String getEndereco() {
23        return endereco;
24    }
25
26    public void setEndereco(String endereco) {
27        this.endereco = endereco;
28    }
29
30    public Date getDataNascimento() {
31        return dataNascimento;
32    }
33
34    public void setDataNascimento(Date dataNascimento) {
35        this.dataNascimento = dataNascimento;
36    }
37
38    public String getSexo() {
39        return sexo;
40    }
41
42    public void setSexo(String sexo) {
43        this.sexo = sexo;
44    }
45
46    public String getCpf() {
47        return cpf;
48    }
49
50    public void setCpf(String cpf) {
51        this.cpf = cpf;
```

```
52 }  
53  
54 public String getTelefone() {  
55     return telefone;  
56 }  
57  
58 public void setTelefone(String telefone) {  
59     this.telefone = telefone;  
60 }  
61  
62 public String getCelular() {  
63     return celular;  
64 }  
65  
66 public void setCelular(String celular) {  
67     this.celular = celular;  
68 }  
69  
70 }
```

Listagem 13 - Código da classe Cliente.

```

1 package br.ufrn.imd.controllers;
2
3 import javax.faces.application.FacesMessage;
4 import javax.faces.bean.ManagedBean;
5 import javax.faces.bean.SessionScoped;
6 import javax.faces.context.FacesContext;
7
8 import br.ufrn.imd.dominio.Cliente;
9
10 /**
11  * MBean para cadastro de cliente.
12  * @author itamir.filho
13  */
14 @ManagedBean
15 @SessionScoped
16 public class ClienteMBean {
17     private Cliente cliente;
18     public ClienteMBean() {
19         cliente = new Cliente();
20     }
21     public String cadastrar() {
22         //validando se foi informado um telefone/celular para contato.
23         if(cliente.getTelefone().equals("") && cliente.getCelular().equals("")){
24             FacesMessage msg = new FacesMessage("Para efetivar o cadastro, é necessário informar um
25             msg.setSeverity(FacesMessage.SEVERITY_ERROR);
26             FacesContext.getCurrentInstance().addMessage("", msg);
27             return "/form.jsf";
28         }
29         return "/sucesso.jsf";
30     }
31     public Cliente getCliente() {
32         return cliente;
33     }
34     public void setCliente(Cliente cliente) {
35         this.cliente = cliente;
36     }
37 }

```

Listagem 14 - Código da classe ClienteMBean.

Observe que nesse controller, ClienteMBean, estamos implementando a validação imperativa dentro do método **cadastar**. Partindo disso, vamos criar o validador customizado para CPF, apresentado na Listagem 15, no pacote **br.ufrn.imd.validadores**.

```
1 package br.ufrn.imd.validadores;
2
3 import javax.faces.application.FacesMessage;
4 import javax.faces.component.UIComponent;
5 import javax.faces.context.FacesContext;
6 import javax.faces.validator.FacesValidator;
7 import javax.faces.validator.Validator;
8 import javax.faces.validator.ValidatorException;
9
10 /**
11  * Validador de CPF.
12  * @author itamir.filho
13  */
14 @FacesValidator("validadorCPF")
15 public class ValidadorCPF implements Validator {
16
17     @Override
18     public void validate(FacesContext context, UIComponent component,
19         Object value) throws ValidatorException {
20         boolean valido = false;
21         String cpf = value.toString();
22         // validando o formato do CPF.
23         try {
24             Long.parseLong(cpf.replaceAll("[\\D]+", ""));
25         } catch (Exception e) {
26             FacesMessage msg = new FacesMessage("CPF inválido: CPF não pode conter letras.");
27             msg.setSeverity(FacesMessage.SEVERITY_ERROR);
28             throw new ValidatorException(msg);
29         }
30
31         // validando a existência do CPF.
32         int soma = 0;
33         if (cpf.length() == 11) {
34             for (int i = 0; i < 9; i++) {
35                 soma += (10 - i) * (cpf.charAt(i) - '0');
36             }
37             soma = 11 - (soma % 11);
38             if (soma > 9)
39                 soma = 0;
40             if (soma == (cpf.charAt(9) - '0')) {
41                 soma = 0;
42                 for (int i = 0; i < 10; i++) {
43                     soma += (11 - i) * (cpf.charAt(i) - '0');
44                 }
45                 soma = 11 - (soma % 11);
46                 if (soma > 9)
47                     soma = 0;
48                 if (soma == (cpf.charAt(10) - '0')) {
49                     valido = true;
50                 }
51             }
52         }
53     }
54 }
```

```
52     }
53     if (!valido) {
54         FacesMessage msg = new FacesMessage(" CPF inválido: Esse CPF não existe.");
55         msg.setSeverity(FacesMessage.SEVERITY_ERROR);
56         throw new ValidatorException(msg);
57     }
58 }
59 }
```

Listagem 15 - Código da classe ValidadorCPF.

Por fim, vamos às views. Teremos duas páginas JSF (xhtml) nessa nossa aplicação. A primeira, denominada **form.xhtml** e apresentada na Listagem 16, é o formulário para obtenção dos dados do cliente. A segunda, denominada **sucesso.xhtml** e apresentada na Listagem 17, é a página de sucesso que apresenta os dados capturados pelo formulário. Veja na Listagem 16 a presença da Tag **validator** para ligarmos a validação do campo de CPF ao nosso validador de CPF.

```
1 <html xmlns="http://www.w3.org/1999/xhtml"
2   xmlns:h="http://java.sun.com/jsf/html"
3   xmlns:f="http://java.sun.com/jsf/core">
4
5   <h:head>
6     <title>Cadastro de Cliente</title>
7   </h:head>
8
9   <h:body>
10    <h:messages/>
11    <h:form>
12      <table>
13        <tr>
14          <td> Nome:* </td>
15          <td> <h:inputText value="#{clienteMBean.cliente.nome}" size="50"
16            required="true" requiredMessage="Nome: Campo obrigatório." /> </td>
17        </tr>
18        <tr>
19          <td> CPF:* </td>
20          <td>
21            <h:inputText value="#{clienteMBean.cliente.cpf}" size="12" maxlength="11" required=
22              requiredMessage="CPF: Campo obrigatório.">
23              <f:validator validatorId="validadorCPF" />
24            </h:inputText>
25          </td>
26        </tr>
27        <tr>
28          <td> Data de nascimento:* </td>
29          <td>
30            <h:inputText value="#{clienteMBean.cliente.dataNascimento}" size="5"
31              maxlength="10" required="true" requiredMessage="Data de nascimento: Campo ob
32              <f:convertDateTime pattern="dd/MM/yyyy" />
33            </h:inputText>
34          </td>
35        </tr>
36        <tr>
37          <td> Endereço:* </td>
38          <td> <h:inputTextarea value="#{clienteMBean.cliente.endereco}" cols="50" rows="4"
39            required="true" requiredMessage="Endereço: Campo obrigatório." /> </td>
40        </tr>
41        <tr>
42          <td> Sexo:* </td>
43          <td>
44            <h:selectOneRadio value="#{clienteMBean.cliente.sexo}"
45              required="true" requiredMessage="Sexo: Campo obrigatório.">
46              <f:selectItem itemValue="M" itemLabel="Masculino" />
47              <f:selectItem itemValue="F" itemLabel="Feminino" />
48            </h:selectOneRadio>
49          </td>
50        </tr>
51      </table>
```



```

52         <td> Telefone: </td>
53         <td> <h:inputText value="#{clienteMBean.cliente.telefone}" /> </td>
54     </tr>
55     <tr>
56         <td> Celular: </td>
57         <td> <h:inputText value="#{clienteMBean.cliente.celular}" /> </td>
58     </tr>
59     <tr>
60         <td colspan="2" align="center">
61             <h:commandButton value="Cadastrar" action="#{clienteMBean.cadastrar}" />
62         </td>
63     </tr>
64 </table>
65
66     * : Campos obrigatórios.
67
68 </h:form>
69 </h:body>
70 </html>

```

Listagem 16 - Código do formulário (form.xhtml).

```

1 <html xmlns="http://www.w3.org/1999/xhtml"
2 xmlns:h="http://java.sun.com/jsf/html"
3 xmlns:f="http://java.sun.com/jsf/core">
4   <h:head>
5     <title>Cliente cadastrado com sucesso!</title>
6   </h:head>
7
8   <h:body>
9     <table>
10      <tr>
11        <td>Nome: </td>
12        <td><h:outputText value="#{clienteMBean.cliente.nome}" /> </td>
13      </tr>
14      <tr>
15        <td>CPF: </td>
16        <td><h:outputText value="#{clienteMBean.cliente.cpf}" /> </td>
17      </tr>
18      <tr>
19        <td>Endereço: </td>
20        <td><h:outputText value="#{clienteMBean.cliente.endereco}" /> </td>
21      </tr>
22      <tr>
23        <td>Data de nascimento: </td>
24        <td>
25          <h:outputText value="#{clienteMBean.cliente.dataNascimento}">
26            <f:convertDateTime pattern="dd/MM/yyyy" />
27          </h:outputText>
28        </td>
29      </tr>
30      <tr>
31        <td>Sexo: </td>
32        <td>
33          <h:outputText value="#{clienteMBean.cliente.sexo}" />
34        </td>
35      </tr>
36      <tr>
37        <td>Telefone/Celular: </td>
38        <td>
39          <h:outputText value="#{clienteMBean.cliente.telefone}" />
40          <h:outputText value="#{clienteMBean.cliente.celular}" />
41        </td>
42      </tr>
43    </table>
44  </h:body>
45 </html>

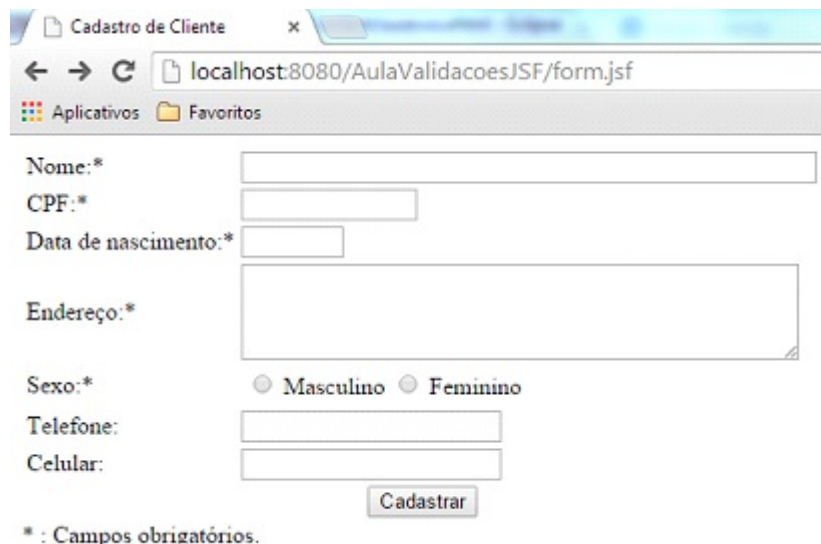
```

Listagem 17 - Código da página de sucesso (sucesso.xhtml).

Após a codificação das classes e páginas JSF apresentadas nas listagens 13,14,15,16 e 17, vamos iniciar o servidor de aplicação Apache Tomcat e acessar nossa aplicação pela URL: <http://localhost:8080/AulaValidacoesJSF/form.jsf>. Realize

alguns testes para verificar o funcionamento da validação. Ao acessar, a página apresentada na Figura 1 é exibida, e ao preenchermos e clicarmos no botão de cadastrar obtemos a página apresentada na Figura 2.

Figura 01 - Página do formulário para cadastro de cliente.



Nome:*

CPF:*

Data de nascimento:*

Endereço:*

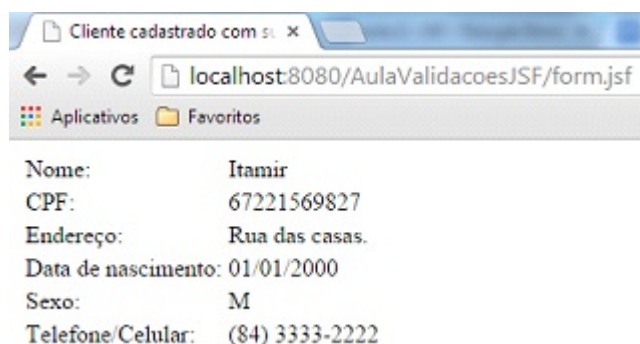
Sexo:* ☐ Masculino ☐ Feminino

Telefone:

Celular:

* : Campos obrigatórios.

Figura 02 - Página de cliente cadastrado com sucesso.



Nome: Itamir

CPF: 67221569827

Endereço: Rua das casas.

Data de nascimento: 01/01/2000

Sexo: M

Telefone/Celular: (84) 3333-2222

E com isso chegamos ao fim da nossa aula. Observe que já conhecemos sobre o framework JSF, suas principais Tags e como realizar validações nas nossas aplicações. Agora você já pode começar a desenvolver aplicações web que podem conter formulários e validações. Na próxima aula, desenvolveremos uma aplicação mais completa para aumentarmos nosso conhecimento sobre esse framework. Estamos progredindo cada vez mais, vamos em frente!

Resumo

Nessa aula, você aprendeu qual a motivação para a realização de validações nos nossos sistemas de informações. Além disso, aprendemos como realizar validações de maneira declarativa, usando as TAGs JSF, e imperativa, utilizando métodos e componentes customizados. Por fim, criamos uma aplicação para explorar essas formas de validações apresentadas. Na próxima aula, iremos desenvolver uma aplicação para formalizar nossos conhecimentos sobre o framework JSF. Até lá e bons estudos!

Referências

New JavaServer Faces 2.2 Feature: The viewAction Component. Disponível em: <<http://www.oracle.com/technetwork/articles/java/jsf22-1377252.html>>. Acesso em: 23 mar. 2015.

Validation in JSF. Disponível em: <<http://incepttechnologies.blogspot.com.br/p/validation-in-jsf.html>>. Acesso em: 23 mar. 2015.

JSF Validator Tags. Disponível em: <http://www.tutorialspoint.com/jsf/jsf_validation_tags.htm>. Acesso em: 23 mar. 2015.

JSF 2 and Bean validation. Disponível em: <<http://www.mastertheboss.com/javaee/jsf/jsf-validation-tutorial?showall=&start=2>>. Acesso em: 23 mar. 2015.

JSF - Custom Validator. Disponível em: <http://www.tutorialspoint.com/jsf/jsf_customvalidator_tag.htm>. Acesso em: 23 mar. 2015.

Custom Validator In JSF 2.0. Disponível em:
<<http://www.mkyong.com/jsf2/custom-validator-in-jsf-2-0/>>. Acesso em: 23 mar.
2015.