FIAP GRADUAÇÃO

DIGITAL BUSINESS ENABLEMENT

Prof. MSc. Rafael Matsuyama

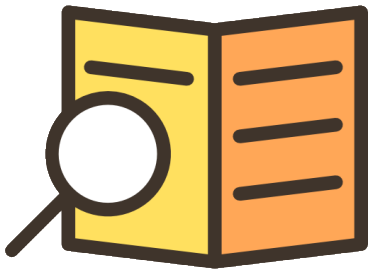
#09 – JSF VALIDAÇÕES E INTERNACIONALIZAÇÃO

 *profrafael.matsuyama@fiap.com.br*  *rafaelmatsuyama*

PERCURSO

- Java Application
- Padrões de Projetos e Frameworks
- SOA e Web Services
- Web Services SOAP
- Web Services Restful
- JSF - Introdução
- JSF – Navegação e Template
- JSF – Validações e Internacionalização

#08 - AGENDA



- Validação
- Mensagens de erros
- Customizando as mensagens de erros
- Validador customizado
- Internacionalização

VALIDAÇÃO

- São responsáveis por **validar** os **campos** utilizados nos formulários;
- As validações sempre **acompanham uma resposta de saída**;
- O processo de validação pode variar em função da **técnica escolhida;**
- Possui o objetivo de manter a **consistência dos dados persistidos** na aplicação e melhorar na **usabilidade**, **ajudando o usuário** no preenchimento dos formulários;
- O **JSF** possui algumas **tags** e **atributos** que facilitam o processo de **validação**;

QUANTIDADE MÁXIMA DE CARACTERES

- O atributo **maxlength** determina a **quantidade máxima** de **caracteres** do campo;
- Já o atributo **size** define visualmente o tamanho do campo;

```
<h:inputText maxlength="10" size="4" />
```

CAMPO OBRIGATÓRIO

- O atributo **required** define se o **campo é obrigatório** ou não;
- É possível determinar uma **mensagem de erro** através do atributo **requiredMessage**, se nenhuma mensagem for definida, será exibida uma **mensagem padrão**;

```
<h:inputText required="true"
  requiredMessage="Campo obrigatório" id="nome"/>
```

INTERVALO E QUANTIDADE

- **<f:validateLongRange/> -** Valida valor numérico inteiro;

```
<h:inputText value="#{clienteBean.cliente.idade}">
  <f:validateLongRange minimum="18" maximum="120"/>
</h:inputText>
```

- **<f:validateDoubleRange/>** - Valida valor numérico real;

```
<h:inputText value="#{produtoBean.produto.preco}">
  <f:validatedDoubleRange minimum="10.50" maximum="10.60"/>
</h:inputText>
```

- **<f:validateLength/> -** Valida String;

```
<h:inputText value="#{clienteBean.cliente.nome}">
  <f:validatedLength minimum="3" maximum="50"/>
</h:inputText>
```

EXPRESSÃO REGULAR

- Podemos realizar validações utilizando **expressão regular;**

```
<h:inputText value="#{cadastroBean.email}">
  <f:validatedRegex pattern="[A-Z].*"/>
</h:inputText>
```

Mais informações sobre **Regex**: http://guia-er.sourceforge.net/

MENSAGENS DE ERRO

- Precisamos exibir as **mensagens de erro** para o usuário, para isso, o JSF possui um **componente** que exibe **todas as mensagens** e um **componente** de mensagem **para um campo específico;**

Mensagem para todos os campos:

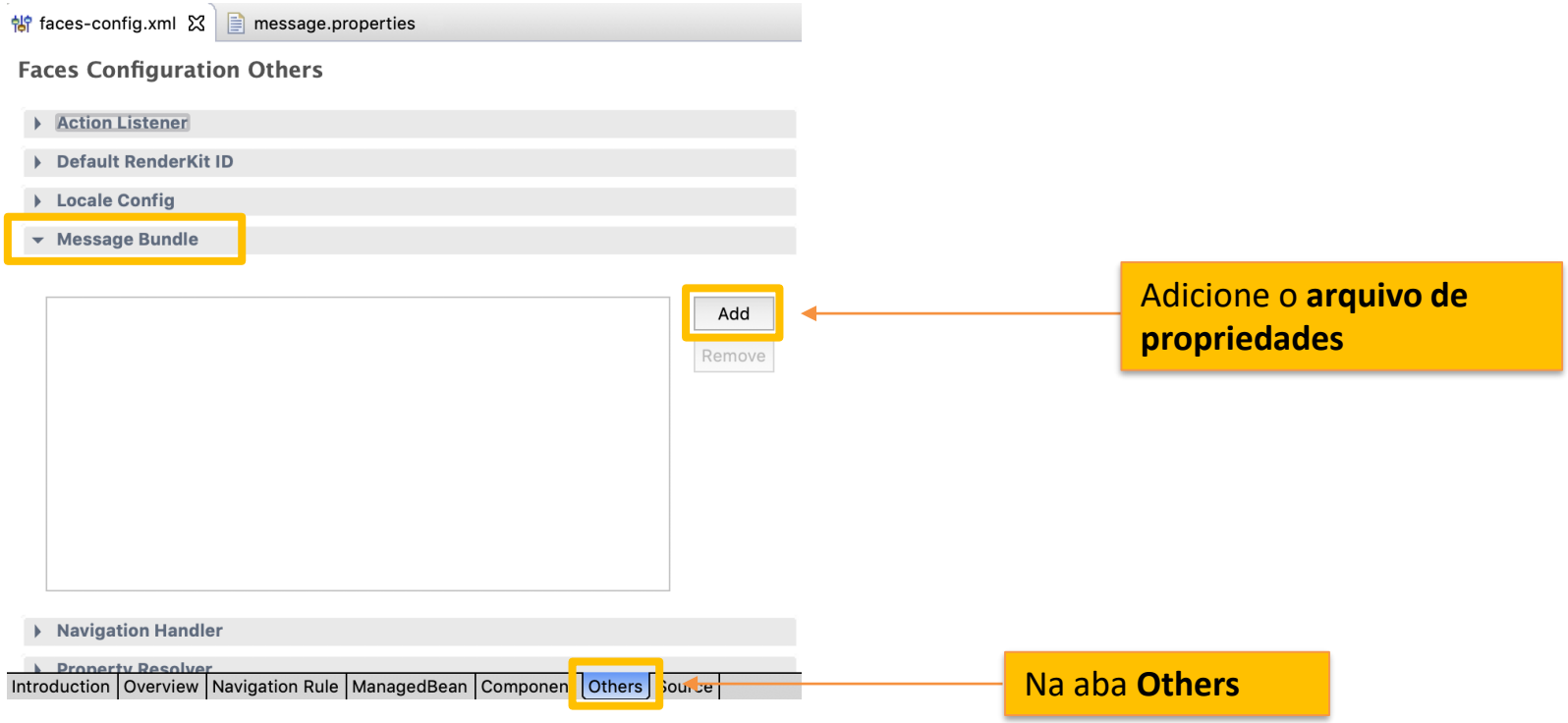
```
<h:messages/>
```

Mensagem para cada campo:

```
<h:inputText required="true"
  requiredMessage="Campo obrigatório" id=”senha“/>

<h:message for=“senha”/>
```

CUSTOMIZANDO AS MENSAGENS DE ERRO

- Podemos criar um **arquivo de propriedades** para definir as nossas próprias **mensagens de erro**;
- Devemos configurar esse arquivo no **faces-config.xml** como um **message bundle**;



CUSTOMIZANDO AS MENSAGENS DE ERRO

- O arquivo de **propriedades** é constituído por **chave** e **valor**, onde a **chave** é definido pelo **framework** e o **valor** será a **mensagem customizada**;

```
javax.faces.validator.LengthValidator.MINIMUM=Valor digitado foi inferior ao mínimo
javax.faces.component.UIInput.REQUIRED=Valor Obrigatório
javax.faces.validator.RegexValidator.NOT_MATCHED=Padrão Inválido
```

VALIDADOR CUSTOMIZADO

- Podemos criar um **método** no **Managed Bean** e implementar uma **validação customizada**, de acordo com as regras de negócio:

Java

```
public void validaDado(FacesContext context, UIComponent component,   Object value)
                                        throws ValidatorException{
   String valor = value.toString();
   if (!valor.startsWith(“A") { //* qualquer validação lógica
     throw new ValidatorException(new FacesMessage("Deveria começar com A"));
   }
}
```

XHTML

```
<h:inputSecret id=”senha" value="#{loginBean.senha}" validator="{#login.validaDado}"/>
```

CRIANDO UM VALIDADOR

- Um **validador** que pode ser utilizado **por várias páginas;**
- A classe deve:
  - Implementar a interface **javax.faces.validator.Validator;**
  - Ser anotada com **@FacesValidator**;
  - Implementar o método **validate()** da interface **Validator**;

CRIANDO UM VALIDADOR - EXEMPLO

- Crie uma **nova classe** e **implemente** a interface **Validator**;
- Defina um **id** para o **validador** no **@FacesValidator**;

Java

```java
@FacesValidator(value="comecaComMaiscula")
public class ValidadorLetraMaiscula implements Validator {

  public void validate(FacesContext context, UIComponent component, Object value) {
    String valor = value.toString();
    if (!valor.matches("[A-Z].*")) {
      throw new ValidatorException(new FacesMessage("Começar com maiscula"));
    }
  }
}
```

XHTML

```xml
<h:inputSecret id="txtSenha" value="#{loginBean.usuario.senha}">
  <f:validator validatorId="comecaComMaiscula"></f:validator>
</h:inputSecret>
```

INTERNACIONALIZAÇÃO

INTERNACIONALIZAÇÃO

- Capacidade da sua aplicação em funcionar com **diferentes idiomas**;
- Trabalho de tradução é chamado de **localização**;
- Necessidade é consequência de um **mundo globalizado;**
- Padrão de nomenclatura segue o modelo **I18N**;
- Melhor abordagem é processar **localização** em **tempo de execução;**
- Conhecimento solicitado na prova de arquiteto Java;

METAS DO I18N

- Minimizar **impacto na programação**;
- Usar mecanismo **genérico e flexível**;
- Manter **separação** entre **código** e **dados** dependentes de língua;
- Facilitar a **contribuição de tradutores**;
- Permitir seleção da **língua em tempo de execução**;

LOCALE

- Conjunto de informações que descreve os **formatos** e mensagens de uma determinada **língua e/ou país**;
- **Processos afetados:**
  - Entrada e saída de dados;
  - Ordenação de palavras;
  - Mensagens;
  - Formatos de data, hora, números, medidas e etc..

IDENTIFICAÇÃO DOS CÓDIGOS

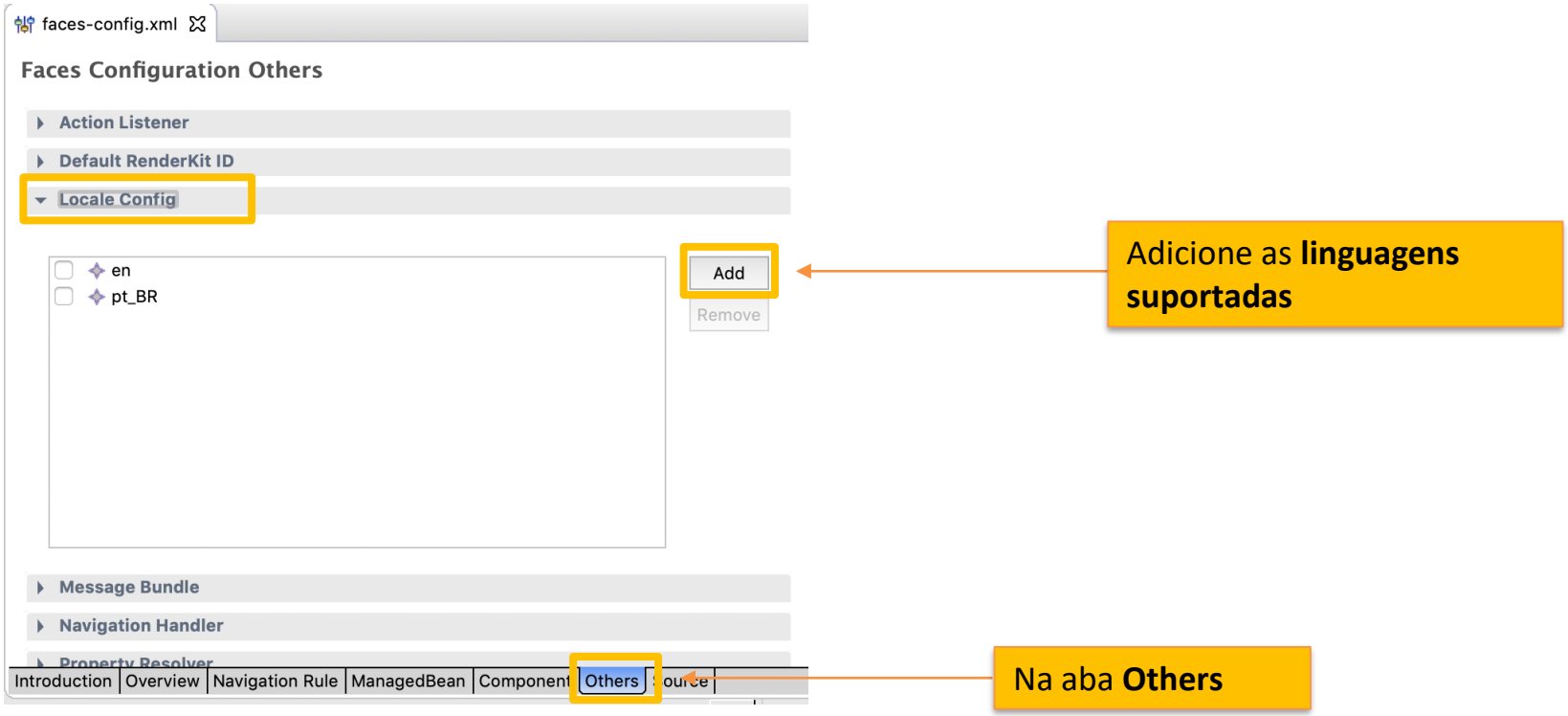
- Código de língua (ISO 639):
  - **pt = português, en = inglês, fr = francês;**
  - http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt
- Código de território (ISO 3166):
  - **BR = Brasil, CA = Canadá, BE = Bélgica;**
  - http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html
- Código (formação) de um locale:
  - **pt_BR, en_CA, fr_BE;**

PROCESSO DE LOCALIZAÇÃO

- Informação do **idioma** é enviada **do browser**;
- **JSF** inicia o processo de **busca na árvore de idiomas** para identificar o mais adequado, exemplo: pr_BR > pt > default;
- **Arquivo de sentenças** definido como .properties. No message-bundle fica contido a informação do diretório e nome do arquivo de mensagem;

IMPLEMENTAÇÃO

- Vamos criar **um arquivo de propriedade para cada linguagem**;
- Adicionar esse arquivo no **message bundle** do **faces-config.xml;**
- No **faces-config.xml** precisamos configurar as **linguagens** suportadas pelo sistema:



ARQUIVO PROPERTIES

- Os arquivos devem possuir as **mesmas chaves**, com a **tradução de cada sentença;**

mensagem_pt_BR.properties

```
javax.faces.validator.LengthValidator.MINIMUM=Valor digitado foi inferior ao minimo
javax.faces.component.UIInput.REQUIRED=Valor Obrigatorio
welcome=Ola {0}!
user-login=Login de Acesso
user-password=Senha
```

mensagem_en.properties

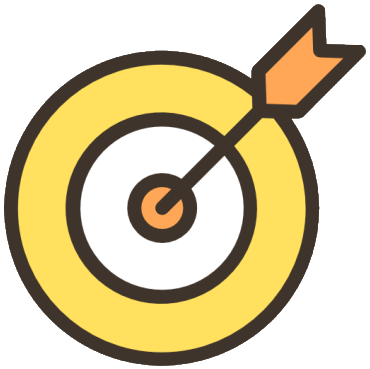
```
welcome=Hello {0}!
user-login=Username
user-password=Password
```

PÁGINA XHTML

- **Carregue o arquivo na página** e defina um valor para a variável;
- Depois utilize a **variável** com as **chaves** definidas no arquivo de propriedades;

```
<f:loadBundle basename="br.com.fiap.bundle.mensagem" var="msgs"/>

<h:outputFormat value="#{msgs.welcome}">                    → welcome=Ola {0}!
  <f:param value="Thiago Yama" />
</h:outputFormat>

<h:outputText value="#{msgs.user-login}"/>                  → user-login=Login de Acesso

<h:inputText required="true" id="nome"
   value="#{fornecedorHandler.fornecedor.nome}">
   <f:validateLength minimum="5"/>
</h:inputText>          → javax.faces.validator.LengthValidator.MINIMUM=Valor digitado foi inferior ao minimo

<h:outputText value="#{msgs.user-password}"/>               → user-password=Senha
```

VOCÊ APRENDEU...



- Realizar **validações de valores** dos campos de formulário;
- **Exibir** as **mensagens de erros**;
- Customizar as **mensagens de erros**;
- Criar um **validador personalizado** e **reutilizável**;
- **Internacionalizar** a aplicação em diferentes idiomas;

"A lógica pode levar de um ponto A a um ponto B.
A imaginação pode levar a qualquer lugar"
Albert Einstein