Lab 2 – Iniciando com JSF

Neste laboratório iremos aprender como criar paginas JSF, utilizar managed beans,

Ao longo do desenvolvimento dos laboratórios de framework você também verá o uso do maven toda vez que for utilizar alguma dependência.

Exercícios

Exercício 1: Criar primeira pagina JSF - Hello Word!

Exercício 2: Usando Managed Beans e escopo de aplicação.

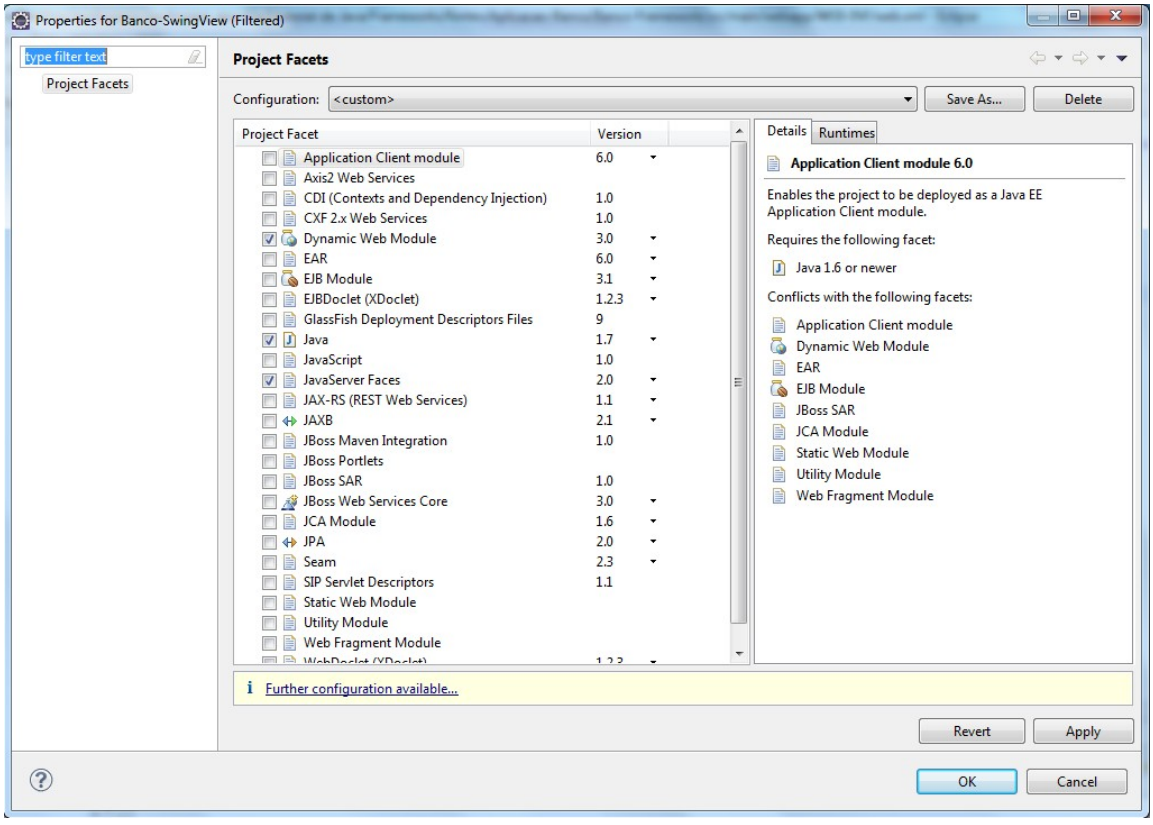
Exercício 3: Criando conversores e validadores.

Exercício 4: Criar regra de navegação em JSF e utilizar facelets para criar um template.

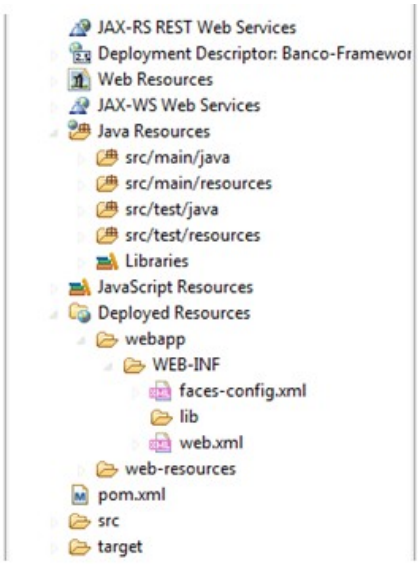
Exercício 5: Internacionalização de conteúdo.

Exercício 1 –Criar primeira página JSF - Hello World.

1. Crie um **módulo maven** para colocar os fontes desse laboratório com o nome de *Lab-Framework.*
2. Tendo o projeto Lab-Framework criado , configure o JSF no mesmo. Clique com **botão direito em cima do projeto > Configure > Add JSF Compabilities...** depois marque e configure as opções de acordo com a imagem abaixo.



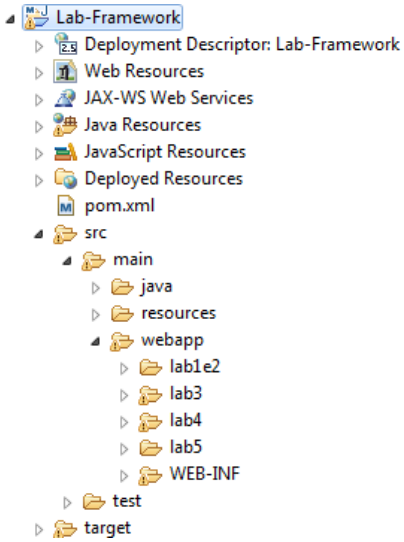
3. O passo anterior irá criar uma estrutura de projeto compatível com a do JSF, criando o *web.xml* e o *faces-config.xml* dentro da pasta WEB-INF.



4. Agora adicione as dependências necessárias do JSF no *pom.xml* do projeto Lab-Framework.

```
<dependencies>

	<dependency>
		<groupId>threeway.projeto.service</groupId>
		<artifactId>Banco-Service</artifactId>
		<version>0.0.1-SNAPSHOT</version>
	</dependency>

	<dependency>
		<groupId>com.sun.faces</groupId>
		<artifactId>jsf-api</artifactId>
		<version>2.2.1</version>
	</dependency>

	<dependency>
		<groupId>com.sun.faces</groupId>
		<artifactId>jsf-impl</artifactId>
		<version>2.2.1</version>
	</dependency>

	<dependency>
		<groupId>javax</groupId>
		<artifactId>javaee-api</artifactId>
		<version>6.0</version>
	</dependency>

</dependencies>
```

5. Crie uma página XHTML dentro de **src/main/webapp/lab1e2** com o nome de **helloWorld.xhtml.**



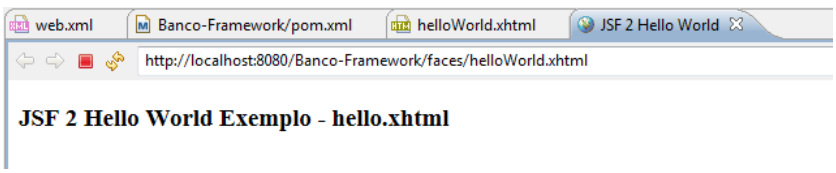
6. Modifique a página **helloWorld.xhtml** de acordo com o código abaixo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h=" http://java.sun.com/jsf/html”>

<h:head>
      <title>JSF 2 Hello World</title>
</h:head>

<h:body>
      <h3>JSF 2 Hello World Exemplo - hello.xhtml</h3>
</h:body>
</html>
```

O código acima cria uma página simples, com um título que vai ser o nome da página e uma frase como conteúdo.

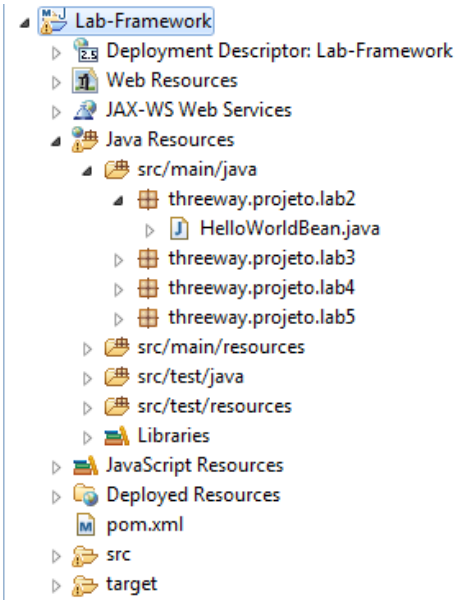
Lembre-se de quando usar os componentes do JSF tem que declarar suas tagLibs correspondentes.

7. Execute a aplicação no servidor Tomcat e acesse pela url: http://localhost:8080/Lab-Framework/faces/helloWorld.xhtml



Exercício 2 –Usando Managed Beans e escopo de aplicação

1. Crie a classe **HelloWorldBean.java** dentro de **src/main/java/threeway/projeto/lab2/**



2. Modifique a classe **HelloWorldBean.java** que acabamos de criar de acordo com o código abaixo:

```
import java.io.Serializable;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;

@ManagedBean
@ViewScoped
public class HelloWorldBean implements Serializable {

	private static final long serialVersionUID = 6949827676782977015L;

	private String nome;

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

}
```

Ao utilizar a anotação *@ManagedBean*, a classe se torna um bean.

Utilizamos o escopo View (anotação @ScopedView) para que cada vez que o usuário acesse a página cria-se uma nova instancia desse Managed Bean.

Para que sua página JSF acesse os atributos de um Managed Bean e necessário que tenha criado os métodos getters e setters desse atributo.

3. Modifique a página **helloWorld.xhtml** criada anteriormente de acordo com o código abaixo, para acesso ao bean de escopo View**.**

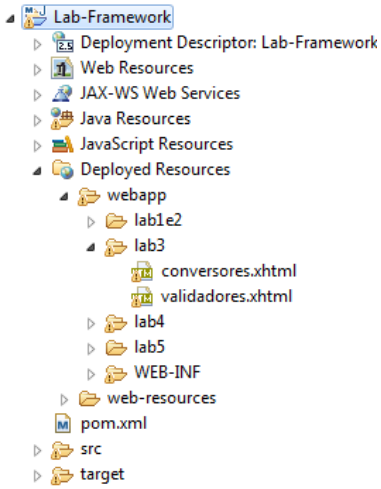
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:h=" http://java.sun.com/jsf/html" >

<h:head>
        <title>JSF 2 Hello World</title>
</h:head>

<h:body>
        <h3>JSF 2 Hello World Exemplo - hello.xhtml</h3>
        <h:form id="form">
                <h:inputText value="#{helloWorldBean.nome}" />
                <h:commandButton value="Bem Vindo" >
                        <f:ajax execute="form" render="form" />
                </h:commandButton>
                <br/><br/>
                <h:outputLabel rendered="#{helloWorldBean.nome != isEmpty}" value="Bem vindo #{hel-
loWorldBean.nome}"></h:outputLabel>
        </h:form>
</h:body>
</html>
```

Veja que adicionamos um formulário (tag <h:form>) que acessará os atributos do bean criado, dando valores e recuperando-os com o comando ***"#{nomeDoBean.atributo}"***.

4. Reinicie servidor Tomcat e acesse a aplicação.

Exercício 3 –Criando Conversores e validadores

1. Crie as páginas XHTML **conversores.xhtml** e **validadores.xhtml** dentro do pacote webapp/lab3/.



2. Crie a classe **ConversoresBean.java** para ser um *Managed Bean*. Basta adicionar a tag *@ManagedBean.*

```
import java.io.Serializable;
import java.util.Date;
import javax.faces.bean.ManagedBean;
import javax.faces.view.ViewScoped;

@ManagedBean(name="conversoresBean")
@ViewScoped
public class ConversoresBean implements Serializable {

	private static final long serialVersionUID = -9003817646964514 48L;

	private Date dataNascimento;
	private Integer peso;
	private String celsiusToFahrenheit;

	public Date getDataNascimento() {
		return dataNascimento;
	}

	public void setDataNascimento(Date dataNascimento) {
		this.dataNascimento = dataNascimento;
	}

	public Integer getPeso() {
		return peso;
	}

	public void setPeso(Integer peso) {
		this.peso = peso;
	}

	public String getCelsiusToFahrenheit() {
		return celsiusToFahrenheit;
	}

	public void setCelsiusToFahrenheit(String celsiusToFahrenheit) {
		this.celsiusToFahrenheit = celsiusToFahrenheit;
	}
}
```

Você pode definir o nome do Managed Bean atribuindo o valor que você queira ao nome, esse nome vai ser para acesso de suas páginas xhtml.

3. Crie a classe **CelsiusToFahrenheitConverter.java** para realizar a conversão da entrada do usuário em Fahrenheit.

```
import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.convert.ConverterException;
import javax.faces.convert.FacesConverter;

@FacesConverter("celsiusToFahrenheitConverter")
public class CelsiusToFahrenheitConverter implements Converter {

	@Override
	public Object getAsObject(FacesContext context, UIComponent component, String value) {

		Float resultado = 0F;
		try {
			Float celsius = Float.parseFloat(value);
			resultado = ( celsius * 9 / 5 ) + 32;

		} catch (Exception e) {

			FacesMessage msg = new FacesMessage("Erro de Conversão em celsiusToFahrenheit-
Converter", "Entrada inválida, tente novamente.");
			msg.setSeverity(FacesMessage.SEVERITY_ERROR );

			throw new ConverterException(msg);
		}
		return resultado;
	}
	@Override
	public String getAsString(FacesContext context, UIComponent component, Object value) {
		return value.toString();
	}
}
```

O código acima irá converter os dados que o usuário passar no formulário, que nesse caso será uma conversão de celsius para fahrenheit.
Usaremos o nome “celsiusToFahrenheitConverter” para identificar o nosso converter personalizado.

Utilize a anotação @FacesConverter para determinar que sua classe será um converter, você também pode optar por implementar no web.xml.

Faça com que sua classe implemente a interface Converter, e implemente os métodos.

Caso ocorra algum imprevisto no código lance a exceção do tipo ConverterException.

4. Com as classes "Controle" prontas, modifique a página **conversores.xhtml** de acordo com o código a seguir. O cabeçalho é o mesmo usado na página **helloWorld.xhtml** criada anteriormente.

```
<h:body>
	<h3>Conversores Padrão</h3>

	<h:form id="formPadrao">
		<h:panelGrid columns="3" cellpadding="10" >

			<h:outputLabel value="Informe a Data de Nacimento" />
			<h:inputText id="nascimento" >
				<f:convertDateTime type="date"  pattern="dd/MM/yyyy"
					timeZone="America/Sao_Paulo"  />
			</h:inputText>
			<h:message for="nascimento" style="color:red"  />

			<h:outputLabel value="Informe seu Peso" />
			<h:inputText id="peso" >
				<f:convertNumber  integerOnly="true"  />
			</h:inputText>
			<h:message for="peso" style="color:red" />

		</h:panelGrid>

		<h:commandButton value="Utilizar Converter Padrão" />
	</h:form>

	<hr/>

	<h3>Conversores Personalizados</h3>
	<h:form id="formCustom">
		<h:panelGrid columns="3" cellpadding="10">

			<h:outputLabel value="Informe a temperatura em Celsius" />

			<h:inputText id="celsius" value="#{conversoresBean.celsiusToFahrenheit}">
				<f:converter converterId="celsiusToFahrenheitConverter "  />
			</h:inputText>

			<h:message for="celsius" style="color:red" />

		</h:panelGrid>

		<h:commandButton value="Calcular Utilizando Converter Customizado" >
			<f:ajax execute="formCustom" render="formCustom"/>
		</h:commandButton>

	</h:form>
</h:body>
```

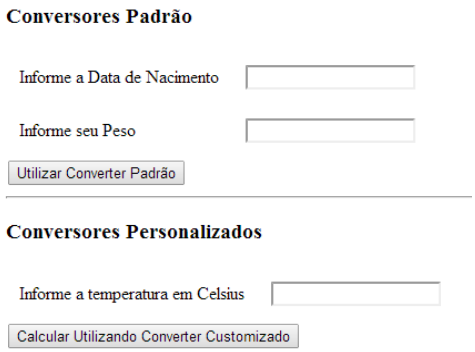
Criamos dois formulários, um para demonstrar o uso de um converter padrão do JSF e outro para demonstrar o uso do converter personalizado que criamos.

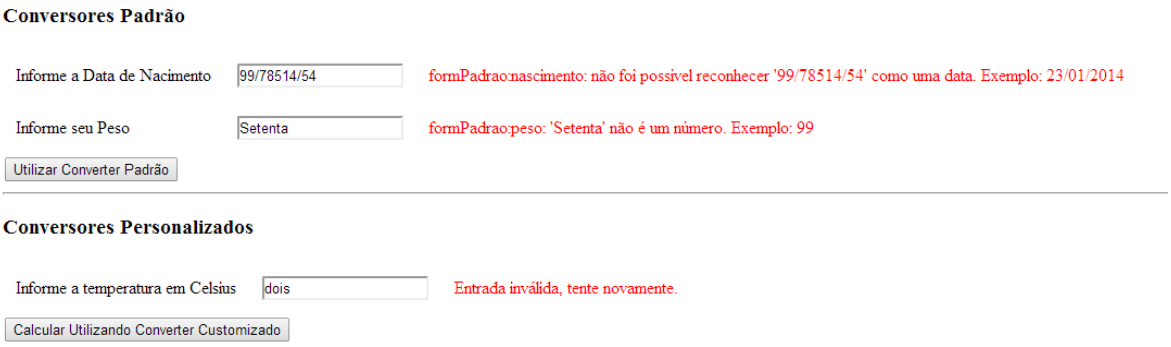
Os conversores padrão do JSF tem suas próprias Tags e próprios atributos específicos.

Para utilização de um converter personalizado terá de ser informado o nome que foi atribuído dentro da anotação @FacesConverter.

Implementamos um botão que envia a requisição para o servidor, e atualiza a página/formulário utilizando ajax.

5. Execute a aplicação e veja o resultado da implementação.





6. Agora vamos trabalhar com validadores. Validadores funcionam da mesma forma que conversores. Crie a classe **ValidadoresBean.java** de acordo com o código abaixo.

```
@ManagedBean(name=”validadoresBean”)
@ViewScoped
public class ValidadoresBean implements Serializable {

	private static final long serialVersionUID = -8703642631958516900L;

	private String atributoObrigatorio;
	private String email;
	private String celsiusToFahrenheit;

	public String getAtributoObrigatorio() {
		return atributoObrigatorio;
	}
	public void setAtributoObrigatorio(String atributoObrigatorio) {
		this.atributoObrigatorio = atributoObrigatorio;
	}
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}
	public String getCelsiusToFahrenheit() {
		return celsiusToFahrenheit;
	}
	public void setCelsiusToFahrenheit(String celsiusToFahrenheit) {
		this.celsiusToFahrenheit = celsiusToFahrenheit;
	}
}
```

7. Criamos três atributos que utilizaremos para validar um formulário. Agora crie uma classe **EmailValidator.java** para validar se o email e válido, siga o código fonte abaixo.

```
@FacesValidator("emailValidator")
```

```
public class EmailValidator implements Validator {
	// Regex que verifica email válido > Caso fique curioso http://regexlib.com/
	private static final String EMAIL_PATTERN = "^[_A-Za-z0-9-]+(\\." + "[_A-Za-z0-9-]+)*@[A-Za-z0-9]+(\\.
[A-Za-z0-9]+)*" + "(\\.[A-Za-z]{2,})$";

	private Pattern pattern;
	private Matcher matcher;

	public EmailValidator() {
		pattern = Pattern.compile (EMAIL_PATTERN);
	}
	@Override
	public void validate(FacesContext context, UIComponent component, Object value)
			throws ValidatorException {

		matcher = pattern.matcher(value.toString());
		if (!matcher.matches()) {
			FacesMessage msg = new FacesMessage
					("Validação de Email falhou.", "Email informado inválido.");

			msg.setSeverity(FacesMessage.SEVERITY_ERROR);

			throw new ValidatorException(msg);

		}

	}

}
```

Utilize a anotação @FacesValidator informar que suas classe é um validator e faça com que sua classe implemente a interface Validator.

Foi utilizado uma expressão regular - REGEX para verificar se a String que o usuário informar obedece a regra da expressão. Para aprender sobre o funcionamento de REGEX aconselho estudar a referencia http://docs.oracle.com/javase/tutorial/essential/regex/

8. Após as classes necessária para lidar com validadores customizados, crie a página **validadores.xhtml** de acordo com o código abaixo;

```
<h:body>
<h3>Validadores Padrão</h3>

	<h:form id="formPadrao">

		<h:panelGrid columns="3">
			<h:outputLabel value="Atributo Obrigatório " />
			<h:inputText id="attrObrigatorio" value="#{validadoresBean.atributoObrigatorio}"
				required="true" />
			<h:message for="attrObrigatorio" style="color:red"/>

			<h:outputLabel value="Informe uma senha: " />
			<h:inputSecret id="senha" value="#{validadoresBean.atributoObrigatorio}"
				required="true" >
				<f:validateLength minimum="5" />
			</h:inputSecret>
			<h:message for="senha" style="color:red" />
		</h:panelGrid>

		<h:commandButton value="Utilizar Validador Padrão" />

	</h:form>

	<hr />

	<h3>Validadores Personalizados</h3>

	<h:form id="formCustom">

		<h:panelGrid columns="3">
			<h:outputLabel value="Entre com email válido: " />
			<h:inputText id="email" value="#{validadoresBean.email}" required="true">
```

```
                        <f:validator validatorId="emailValidator " />
                    </h:inputText>
                    <h:message for="email" style="color:red" />
                </h:panelGrid>

                <h:commandButton value="Utilizar Validador Customizado"/>

            </h:form>

            <hr />

            <h3>Validadores + Conversores</h3>
            <h:form id="formValidConvert">
                <h:panelGrid columns="3" cellpadding="10">
                    <h:outputLabel value="Informe a temperatura em Celsius" />
                    <h:inputText id="celsius" value="#{conversoresBean.celsiusToFahrenheit}">
                        <f:validateLongRange maximum="100" minimum="-50"/>
                        <f:converter converterId="celsiusToFahrenheitConverter "/>
                    </h:inputText>
                    <h:message for="celsius" style="color:red" />
                </h:panelGrid>

                <h:commandButton value="Calcular" >
                    <f:ajax execute="formValidConvert" render="formValidConvert"/>
                </h:commandButton>

            </h:form>
            <h:body>
```

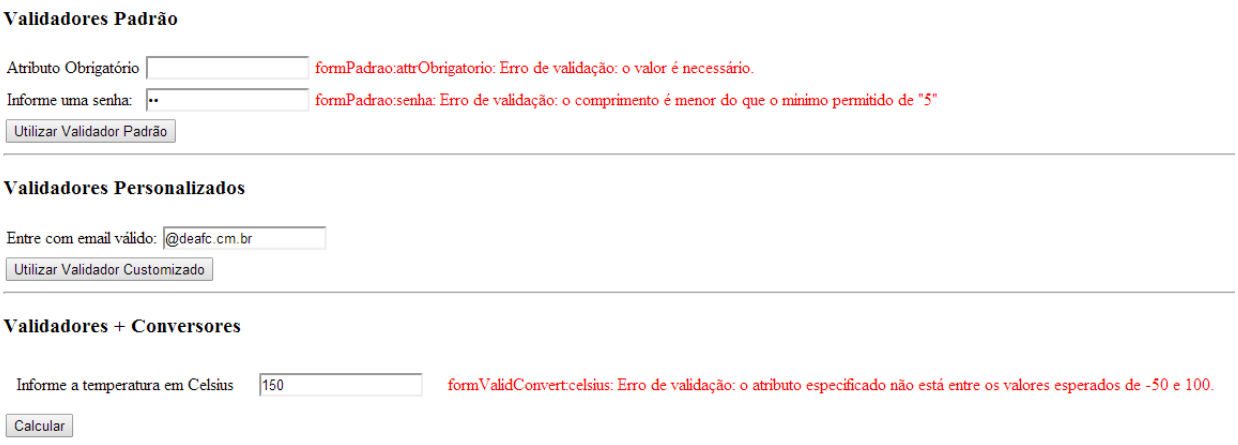
Criamos dois formulários, um para demonstrar o uso de um validator padrão do JSF e outro para demon strar o uso do validator personalizado que criamos.

Os validadores padrão do JSF tem suas próprias Tags e próprios atributos específicos.

Para utilização de um validator customizado terá de ser informado o nome que foi atribuído dentro da anotação @FacesValidator

Em um mesmo campo de entrada do usuário pode usar validators e converters. Fique atento ao ciclo de vida e quando cada um será executado.

9. Execute a aplicação e veja o resultado.



Exercício 4 –Criar regra de navegação em JSF e utilizar facelets para criar um template

1. Para criarmos um template utilizando facelets, crie um arquivo de css com o nome **style.css** onde vamos definir e dividir nossa página. Siga o código abaixo.

```
html,body {
        height: 99%;
        margin: 1px;
        padding: 0;
        border: 0;
}
div {
        margin: 0;
        border: 0;
}
.content {
        display: table;
        width: 100%;
        border-collapse: separate;
        height: 80%;
}
.Col-left {
        display: table-cell;
        width: 15%;
        height: 100%;
}
.Col-center {
        display: table-cell;
        width: 70%;
        height: 100%;
}

.Col-right {
        display: table-cell;
        width: 15%;
        height: 100%;
}

.border {
        border: 1px solid;
        border-radius: 25px;
}
#header,#footer {
        height: 10%;
        position: relative;
        z-index: 1;
}
```

2. Agora crie a página **template.xhtml** seguindo o exemplo abaixo.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:ui=" http://java.sun.com/jsf/facelets" >

<h:head>
        <title>JSF 2 - Facelets</title>
        <link rel="stylesheet" type="text/css"
href="#{facesContext.externalContext.requestContextPath}/lab4/style.css" />
</h:head>

<h:body>
        <div id="header" class="border">
                <h2>Topo da Página</h2>
        </div>

        <div class="content">
                <div id="left" class="Col-left border">
```

```
                        <h2>Lado esquerdo</h2>
                        <ui:include src="menu-esquerdo.xhtml" />
                </div>
                <div id="center" class="Col-center border">
                        <h2>Conteúdo da página</h2>
                        <ui:insert name="conteudo" />
                </div>
                <div id="right" class="Col-right border">
                        <h2>Lado direito</h2>
                </div>
        </div>

        <div id="footer" class="border">
                <h2>Rodapé</h2>
        </div>
</h:body>
</html>
```

Para utilizar Facelets declare a tagLib no inicio da pagina.

Você pode inserir páginas utilizando a tag <ui:include src="pagina.xhtml" />

Definindo seu template você pode usar a tag <ui:insert name="conteudo" /> onde qualquer página que for definida como uma composition e utilizar esse template irá preencher o conteúdo da página.

3. Crie a página **menu-esquerdo.xhtml** pois esta está sendo inserida em seu template na tag <ui:include/>.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:ui="http://java.sun.com/jsf/facelets">

<h:body>
        <h3>Menu esquerdo</h3>
</h:body>
</html>
```

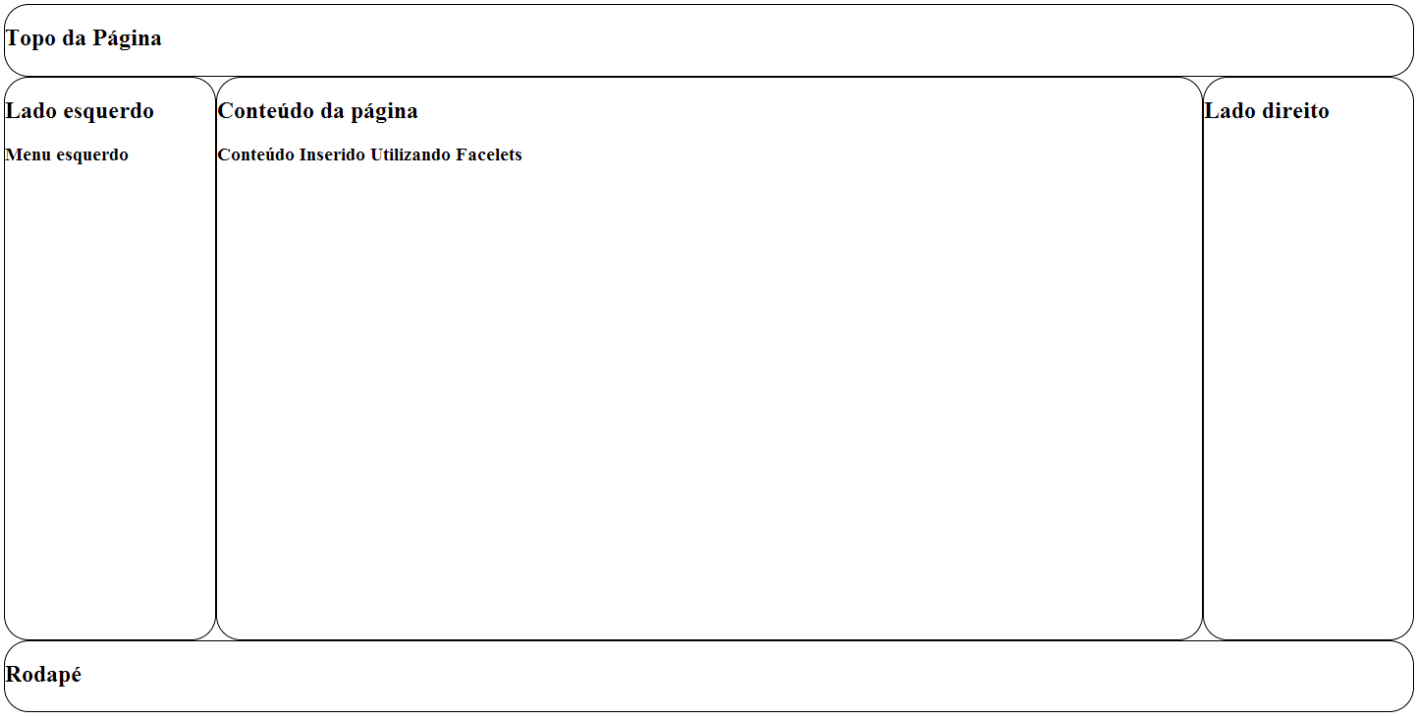
4. Agora crie a página que utilizara seu template, dê o nome de **index.xhtml**.

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:ui="http://java.sun.com/jsf/facelets" template="template.xhtml">

        <ui:define name="conteudo">

                <h3>Conteúdo Inserido Utilizando Facelets</h3>

        </ui:define>

</ui:composition>
```

Para definir que uma página usara um template você tera que definir que ela será uma composition utilizando a tag <ui:composition />

Para definir qual parte do código ira ser adicionada no template, você que coloca entre a tag <ui:define name="conteudo" /> onde o conteúdo será o nome que você definiu na tag <ui:insert />

5. Execute a página **index.xhtml** e você vera o poder do facelets para reaproveitamento de código.



6. Agora, para utilizarmos a navegação do JSF, crie uma página xhtml com o nome de **pagina1.xhtml** que utilize o template.

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:ui="http://java.sun.com/jsf/facelets"
        template="template.xhtml">

        <ui:define name="conteudo">

                <h3>Navegação Página 1</h3>

        </ui:define>

</ui:composition>
```

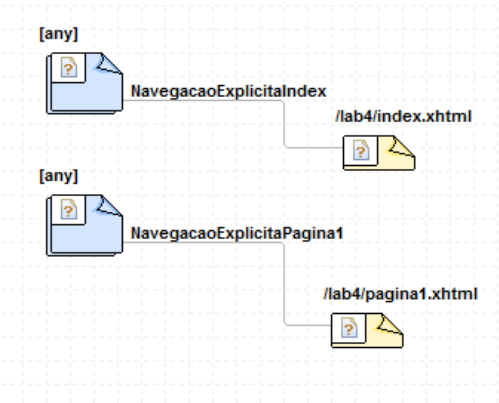
7. Crie a classe **NavegacaoBean.java** dentro de **src/main/java/threeway/projeto/lab4** e modifique de acordo com o código abaixo. Essa classe implementará métodos para navegação dinâmica da página.

```
@ManagedBean(name=”navegacaoBean”)
@ViewScoped
public class NavegacaoBean implements Serializable {

	private static final long serialVersionUID = -7622250974049755899L;

	public String navegacaoDinamicaImplicitaIndex() {

		return "index";
	}
	public String navegacaoDinamicaImplicitaPagina1() {

		return "pagina1";
	}
	public String navegacaoDinamicaExplicitaIndex() {

		return "NavegacaoExplicitaIndex";
	}
	public String navegacaoDinamicaExplicitaPagina1() {

		return "NavegacaoExplicitaPagina1";
	}
}
```

Cada método desse irá retornar o nome de uma das páginas xhtml que criamos anteriormente.

8. Para utilizar a navegação explícita, modifique o código de **faces-config.xml** dentro de WEB-INF adicionando as seguintes regras de navegação.

```
<navigation-rule>
	<navigation-case>
		<from-outcome>NavegacaoExplicitaIndex</from-outcome>
		<to-view-id>/lab4/index.xhtml</to-view-id>
	</navigation-case>
</navigation-rule>
<navigation-rule>
	<navigation-case>
		<from-outcome>NavegacaoExplicitaPagina1</from-outcome>
		<to-view-id>/lab4/pagina1.xhtml</to-view-id>
	</navigation-case>
</navigation-rule>
```

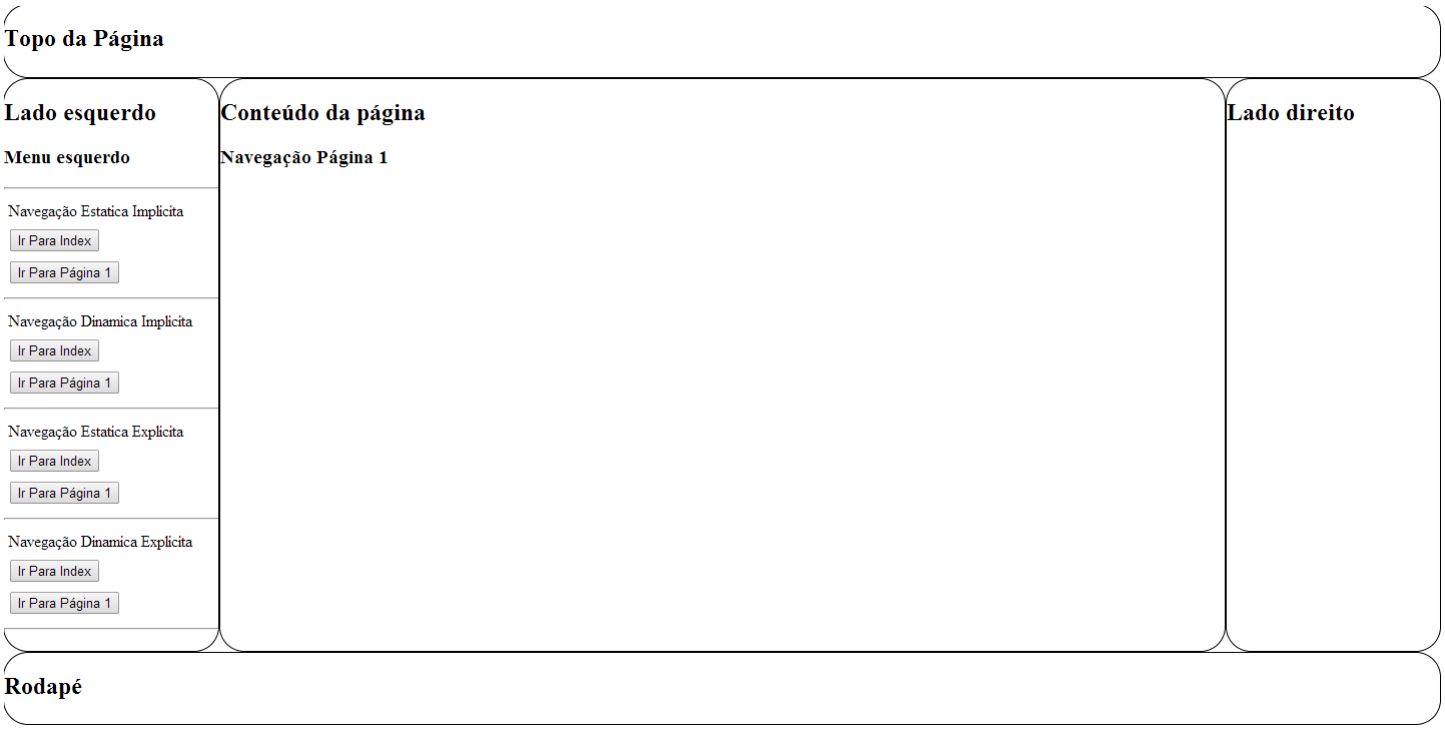


9. Agora modifique a página **menu-esquerdo.xhtml,** criada anteriormente, para utilizar a navegação implícita (dinâmica e estática) e explicita (dinâmica e estática).

```
<h:form>
	<hr/>
	<h:panelGrid cellpadding="2">
		Navegação Estatica Implicita
		<h:commandButton value="Ir Para Index" action="index" />
		<h:commandButton value="Ir Para Página 1" action="pagina1" />
	</h:panelGrid>
	<hr/>
	<h:panelGrid cellpadding="2">
		Navegação Dinamica Implicita
		<h:commandButton value="Ir Para Index" action="#{navegacaoBean.navegacaoDinamicaImplicitaIndex()}" />
		<h:commandButton value="Ir Para Página 1" action="#{navegacaoBean.navegacaoDinamicaImplicitaPagina1()}" />
	</h:panelGrid>
	<hr/>
	<h:panelGrid cellpadding="2">
		Navegação Estatica Explicita
		<h:commandButton value="Ir Para Index" action="NavegacaoExplicitaIndex" />
		<h:commandButton value="Ir Para Página 1" action="NavegacaoExplicitaPagina1" />
	</h:panelGrid>
	<hr/>
	<h:panelGrid cellpadding="2">
		Navegação Dinamica Explicita
		<h:commandButton value="Ir Para Index" action="#{navegacaoBean.navegacaoDinamicaExplicitaIndex()}" />
		<h:commandButton value="Ir Para Página 1" action="#{navegacaoBean.navegacaoDinamicaExplicitaPagina1()}" />
	</h:panelGrid>
	<hr/>
</h:form>
```

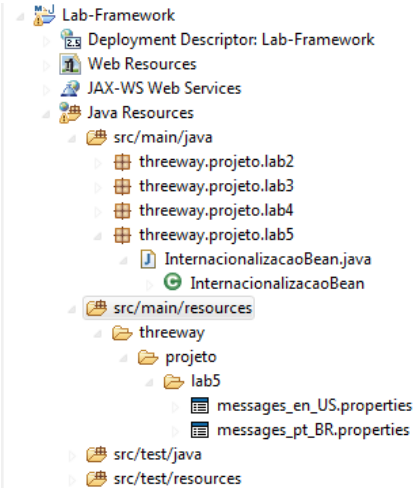
Veja que na navegação dinâmica estamos acessamos os métodos do bean criado, enquanto na navegação estática passamos diretamente o nome da página a ser acessada.

10. Execute a página **index.xhtml**. Pronto você tem um exemplo dos tipos de navegação do JSF.



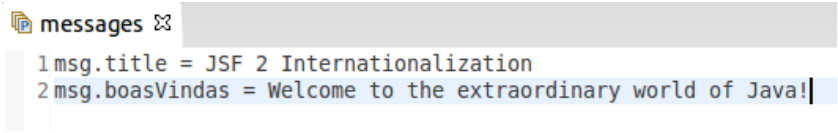
Exercício 5 – Internacionalização de conteúdo.

1. Como estamos utilizando o Maven, colocaremos nossos arquivos **.properties** dentro do seguinte diretório **src/main/resources/threeway/projeto/lab5.** Agora criaremos dois arquivos **properties** como mostra a imagem abaixo. Lembre-se do padrão mencionado na apostila.

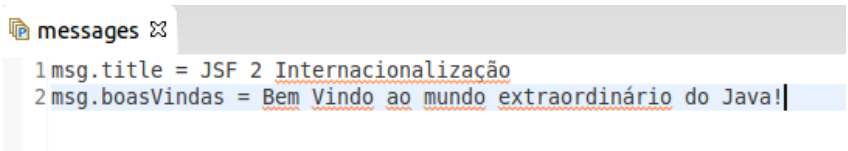


2. Modifique esses arquivos criados criando duas mensagens exemplo conforme a imagem abaixo. Lembre-se que as mensagens deverão ter o mesmo nome correspondendo a valores diferentes.

   Inglês:



   Português:



3. Agora crie a classe **InternacionalizacaoBean.java** onde terá os métodos para mudar o estado do objeto **Locale.** Esse Managed Bean terá o escopo de sessão (anotação @SessionScoped), pois enquanto a sessão existir para um usuário sua escolha estará mantida.

```
@ManagedBean
@SessionScoped
public class InternacionalizacaoBean implements Serializable {

	private static final long serialVersionUID = -4717414476624260 81L;
	private Locale currentLocale = new Locale("pt", "BR");

	public void englishLocale() {

		UIViewRoot viewRoot = FacesContext.getCurrentInstance().getViewRoot();
		currentLocale = Locale.US;
		viewRoot.setLocale(currentLocale);
	}

	public void portugueseLocale() {

		UIViewRoot viewRoot = FacesContext.getCurrentInstance().getViewRoot();
		currentLocale = new Locale("pt", "BR");
		viewRoot.setLocale(currentLocale);
	}

	public Locale getCurrentLocale() {
		return currentLocale;
	}
}
```

4. Modifique o arquivo **faces-config.xml,** que está dentro da pasta WEB-INF, para que ele faça a "leitura" desses arquivos de properties criados.

```
<application>

	<locale-config>
		<default-locale>pt_BR</default-locale>
		<supported-locale>pt_BR</supported-locale>
		<supported-locale>en_US</supported-locale>
	</locale-config>

	<resource-bundle>
		<base-name>threeway.projeto.lab5.messages</base-name>
		<var>msg</var>
	</resource-bundle>

	<message-bundle>
		threeway.projeto.lab5.messages
	</message-bundle>

</application>
```

O caminho para o arquivo **messages** deve ser dado a partir da pasta *resources*.
Adicione o nome da variável a ser utilizada para acessar os valores dos arquivos .properties

dentro da tag *<var/>*.

5. Agora crie a página **internacionalização.xhtml**. Utilize os ícones para as ações de internacionalização.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">

<h:head>
      <title>#{msg['msg.title']}</title>
</h:head>

<h:body>
      <h3>#{msg['msg.title']}</h3>

      <h:form>
            <h:panelGrid columns="2" cellpadding="5">

                  <h:commandLink action="#{internacionalizacaoBean.englishLocale()}" >
                        <h:graphicImage value="/lab5/estados_unidos.png" />
                  </h:commandLink>

                  <h:commandLink action="#{internacionalizacaoBean.portugueseLocale()}" >
                        <h:graphicImage value="/lab5/brasil.png" />
                  </h:commandLink>

            </h:panelGrid>
      </h:form>

      <h2>#{msg['msg.boasVindas']}</h2>

</h:body>
</html>
```

Para acessar o arquivo de messages definido no faces-config.xml utilize o nome da variavél que você atribuiu dentro do atributo <var>nomeVariavel</var>

6. Execute a página para ver o resultado da implementação.