

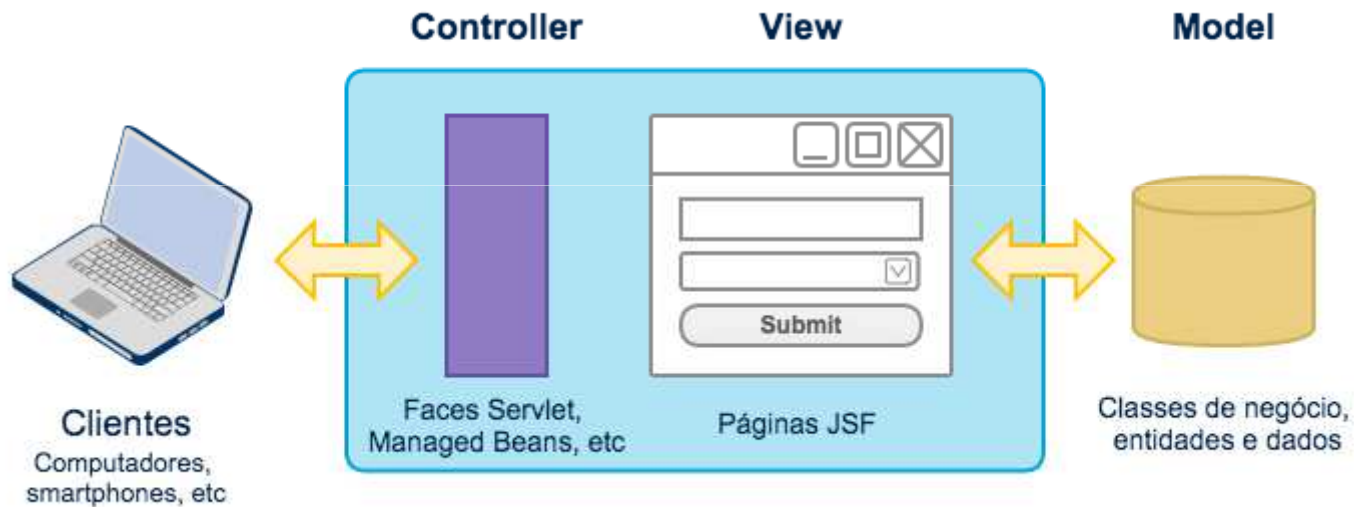
PROGRAMAÇÃO PARA WEB II

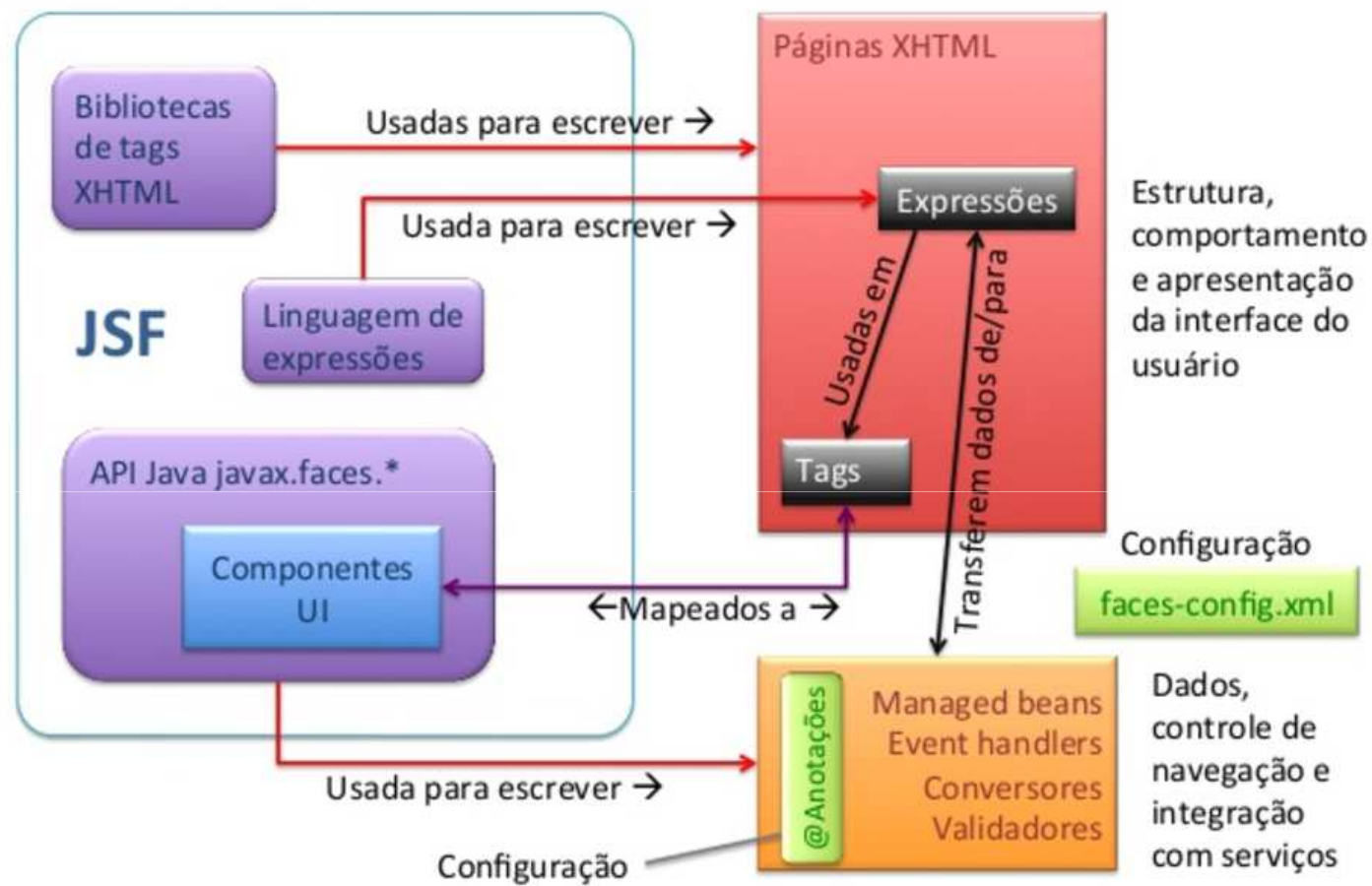
AULA 2

Profa. Silvia Bertagnolli

JSF

MVC — MODEL VIEW CONTROLLER





Fonte: <https://pt.slideshare.net/helderdarocha/tutorial-jsf-20-2012>

COMPONENTES DA ESPECIFICAÇÃO MOJARRA

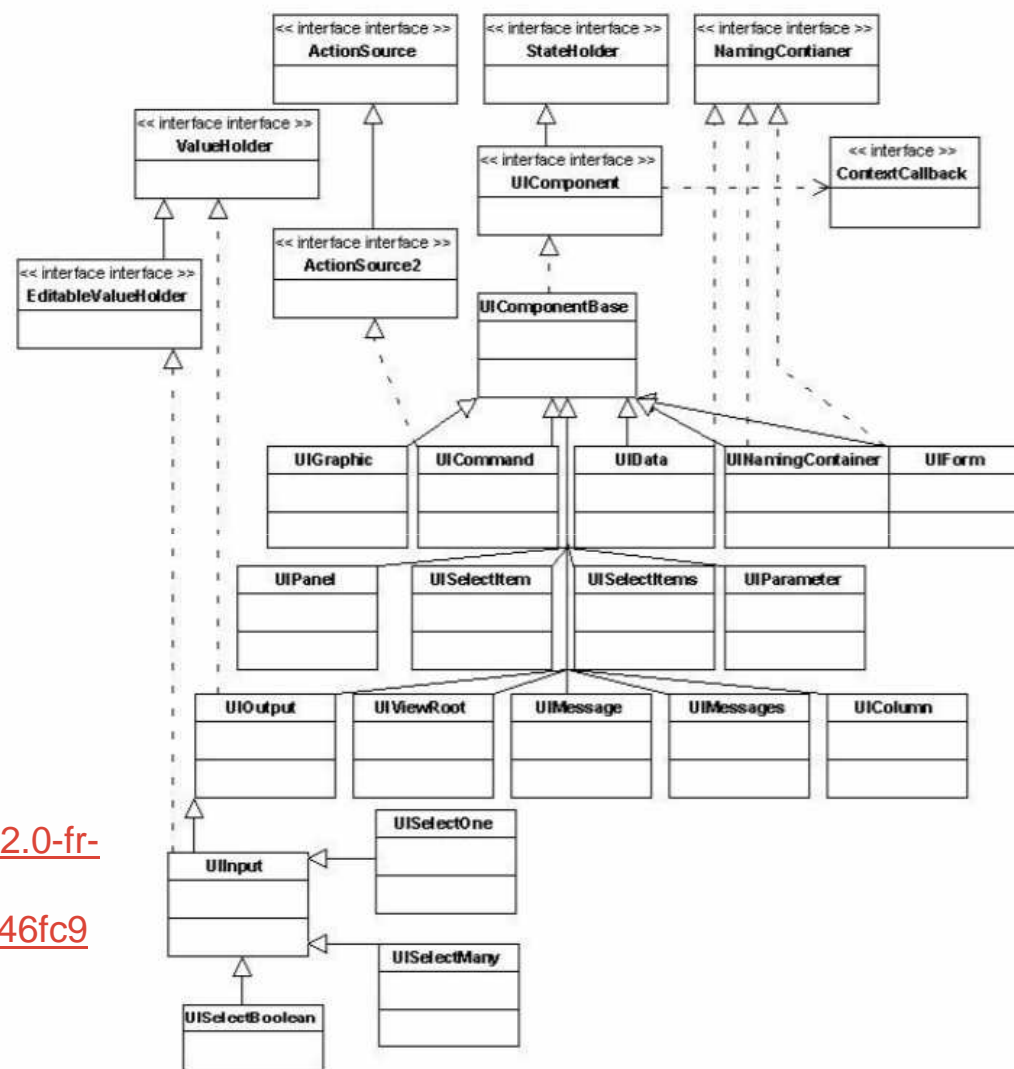
The screenshot shows a web form with the following elements:

- A text input field containing "JSF".
- A password input field with five asterisks "*****".
- A text area containing the text "JSF é component-based.".
- A dropdown menu currently showing "JSF".
- A row of radio buttons with labels: "JSF", "Tapestry", "vaadin", "Wicket", and "GWT". The "JSF" radio button is selected.
- A list box containing the same five framework names: "JSF", "Tapestry", "vaadin", "Wicket", and "GWT".
- A "salva" button.
- A blue underlined link "salva" below the button.

<https://www.caelum.com.br/apostila-java-testes-jsf-web-services-design-patterns/introducao-ao-jsf-e-primefaces/>

<https://javaee.github.io/jaserverfaces-spec/>

HIERARQUIA DOS COMPONENTES



Fonte: http://download.oracle.com/otn-pub/jcp/jsf-2.0-fr-full-oth-JSpec/jsf-2_0-fr-spec.pdf?AuthParam=1501788925_50ae975fc6746fc92a60b543bef78710

Página 138

PRINCIPAIS PACOTES

`java.faces.bean` – anotações para managed beans

`java.faces.component` – componentes gráficos

`java.faces.context` – acesso ao estado da requisição

`java.faces.convert` – conversores de dados

`java.faces.event` – tratamento de eventos

`java.faces.render` – renderização gráfica

`java.faces.validator` – validadores de dados

ESCOPO DOS BEANS

MANAGED BEANS: ESCOPOS

@RequestScoped

Objetos armazenados nesse escopo existem do envio da requisição até o retorno da sua resposta

Escopo usado quando não precisamos guardar os dados entre as requisições dos usuários

Escopo padrão – se você não incluir a anotação informando um escopo estará usando automaticamente @RequestScoped

@SessionScoped

MANAGED BEANS: ESCOPOS

@SessionScoped

Tudo fica disponível enquanto a sessão do usuário estiver ativa

Exemplos de uso:

- 1) Usuário logado
- 2) Última ação realizada para voltar mais rápido para ela

MANAGED BEANS: ESCOPOS

@ApplicationScoped

Tudo que for armazenado no escopo de aplicação permanece enquanto a aplicação estiver executando e é compartilhada entre todos os usuários

Exemplo: guardar usuários online e possibilitar a comunicação entre eles

MANAGED BEANS: ESCOPOS

@ViewScoped

Manter os dados contidos no escopo por quantas requisições forem feitas, desde que sejam todas para a mesma view

Ao trocar de página, usando o método POST, o contexto é zerado. Caso contrário os objetos permanecem na memória do servidor



Defina o escopo da classe UsuarioBean para que os dados do usuário logado fiquem disponíveis para o sistema durante a sua execução

VALIDADORES

VALIDADORES: NATIVOS

Os validadores nativos são validações prontas que existem no JSF:

`f:validateLength`

`f:validateLongRange`

`f:validateDoubleRange`

`f:validateRequired*`

`f:validateRegex`

* gera o mesmo resultado que a propriedade `required` dos componentes JSF



O que cada um dos validadores listados anteriormente faz?

Na página `operacoes_aluno` qual validador foi usado?

COMO CRIAR VALIDADORES?

Para criar um validator é necessário definir uma classe java que implementa `javax.faces.validator.Validator`

Um exemplo prático é validar o Cpf que o usuário forneceu – se ele colocou um valor válido, por exemplo

PASSOS

Passo 1 – Criar a classe de validação

```
public class CpfValidator implements Validator { ... }
```

Passo 2 – Criar o arquivo faces-config

Novo -> Java Server Faces -> Configuração do JSF Faces

Cria o arquivo faces-config na pasta arquivo de configuração

PASSOS

Passo 3 – Registrar o Validador no arquivo de configuração

```
<validator>
```

```
    <validator-id>classes.CpfValidator</validator-id>
```

```
    <validator-class>classes.CpfValidator</validator-class>
```

```
</validator>
```

PASSOS

Passo 4 – Vincular o validator ao componente

```
<h:inputText id="cpf" value="#{alunoBean.aluno.cpf}"  
             required="true" maxLength="11" label="cpf"  
             <f:validator validatorId="classes.CpfValidator" />  
</h:inputText>
```



Vamos analisar a página `operacoes_aluno` e verificar onde o validador criado é usado

Crie um validador para o CEP usado na classe `Aluno`

CONVERSORES

CONVERSORES NATIVOS

JSF define um conjunto de conversores nativos para números e datas

Uma propriedade do tipo `java.lang.Integer` em seu formulário
JSF tem o seu conversor automático
`javax.faces.convert.IntegerConverter`

`f:convertDateTime`

`f:convertNumber`

CONVERSORES NATIVOS: CONVERTDATETIME

Pattern – usado para determinar o padrão que a data será formatada

dateStyle, type, locale e timezone

CONVERSORES NATIVOS: CONVERTDATETIME

`dateStyle` – pode assumir os valores com os respectivos resultados:

- `default` = 21/12/2012
- `short` = 21/12/12
- `medium` = 21/12/2012
- `long` = 21 de Dezembro de 2012
- `full` = Sexta-feira, 21 de Dezembro de 2012

CONVERSORES NATIVOS: CONVERTDATETIME

type

- date = 21/12/2012
- time = 23:59:59
- both = 21/12/2012 23:59:59
- long = 21 de Dezembro de 2012
- full = Sexta-feira, 21 de Dezembro de 2012

CONVERSORES NATIVOS: CONVERTDATETIME

Exemplo:

```
<h:inputText id="data"
    value="#{alunoBean.aluno.dataIngresso}" required="true"
    label="data">
    <f:convertDateTime dateStyle="full" type="both"/>
</h:inputText>
```

CONVERSORES NATIVOS: CONVERTNUMBER

`maxFractionDigits` e `minFractionDigits`

Usados para determinar o máximo e o mínimo de casas decimais que devem ser usadas

Exemplo:

```
<h:outputText value="#{userBean.height}">
```

```
<f:convertNumber maxFractionDigits="2" />
```

```
</h:outputText>
```



Vamos analisar a página `operacoes_aluno` e verificar o funcionamento dos conversores

Quais conversores foram usados?

COMO CRIAR CONVERSORES?

Um conversor é definido com uma classe java que implementa `javax.faces.converter.Converter`

Ele é usado para que os dados informados em um campo no formulário sejam convertidos automaticamente para um objeto e vice-versa

Métodos:

`getAsString()` – converte o objeto para String

`getAsObject()` – converte a String para o formato de um objeto

Link: <http://blog.algaworks.com/conversores-jsf/>

PASSOS

Passo 1 – Criar a classe para Conversão

```
public class TelefoneConverter implements Converter{ ... }
```

Passo 2 – Criar o arquivo faces-config

Novo -> Java Server Faces -> Configuração do JSF Faces

Cria o arquivo faces-config na pasta arquivo de configuração

PASSOS

Passo 3 – Registrar o Conversor no arquivo de configuração

```
<converter>
  <description>Configurando converter para classe
Telefone</description>
  <converter-id>telefoneConverter</converter-id>
  <converter-class>classes.TelefoneConverter</converter-class>
</converter>
```

**OU Passo 3 – Registrar o Conversor usando a anotação
@FacesConverter**

PASSOS

Método `getAsString()`

Retorna o objeto Telefone formatado incluindo o "+" do código internacional

Método `getAsObject()`

Se a String for nula ou vazia é retornada uma referência nula, caso contrário, a String é convertida para somente números e os vincula aos atributos corretos



Analise a classe TelefoneConverter e determine:

O que faz o catch() do método getAsObject()?

Qual o objetivo da classe FacesMessage? Onde e quando ela é usada?

PASSOS

Passo 4 – Vincular o conversor ao componente

```
<h:inputText id="telefone" value="#{alunoBean.aluno.telefone}"  
              required="true" label="telefone">  
    <f:converter converterId="telefoneConverter" />  
</h:inputText>
```

Fonte: <http://blog.algaworks.com/conversores-jsf/>



Crie a classe Nome e modifique a classe Pessoa/Aluno de modo que o nome seja definido usando composição

Crie o converter para o nome usando os passos vistos nos slides anteriores

AJAX

AJAX NO JSF

Ajax é baseado em JavaScript e XML

JSF (2.0 ou superior) possui comunicação com Ajax padrão isso torna transparente as chamadas cliente-servidor sem a necessidade de recarregar a página

Para usar o ajax é necessário:

- 1) Incluir as TAGs - `xmlns:f="http://xmlns.jcp.org/jsf/core">`
- 2) Incorporar ao componente a TAG `<f:ajax>`

AJAX NO JSF

```
<h:selectOneMenu value="#{ajaxMB.estadoSelecioneado}">
  <f:selectItem itemLabel="#{mensagens.ajaxSelecione}"
                itemValue=""/>
  <f:selectItems value="#{ajaxMB.estados}"/>
  <f:ajax execute="@this"
          render="selectCidade selectBairro"
          listener="#{ajaxMB.alterarEstado}" />
</h:selectOneMenu>
```

f:ajax = é responsável pela chamada assíncrona
Cada vez que o conteúdo de selectOneMenu é
alterado o ajax envia o valor definido usando
execute="@this"
@this = componente atual

AJAX NO JSF

```
<h:selectOneMenu value="#{ajaxMB.estadoSelecioneado}">
  <f:selectItem itemLabel="#{mensagens.ajaxSelecione}"
                itemValue=""/>
  <f:selectItems value="#{ajaxMB.estados}"/>
  <f:ajax execute="@this"
          render="selectCidade selectBairro"
          listener="#{ajaxMB.alterarEstado}" />
</h:selectOneMenu>
```

Atributo render indica quais componentes serão atualizados quando a resposta assíncrona for retornada

No caso desse componente selectCidade selectBairro dependem da seleção

AJAX NO JSF

```
<h:selectOneMenu value="#{ajaxMB.estadoSelecioneado}">
  <f:selectItem itemLabel="#{mensagens.ajaxSelecione}"
                itemValue=""/>
  <f:selectItems value="#{ajaxMB.estados}"/>
  <f:ajax execute="@this"
          render="selectCidade selectBairro"
          listener="#{ajaxMB.alterarEstado}" />
</h:selectOneMenu>
```

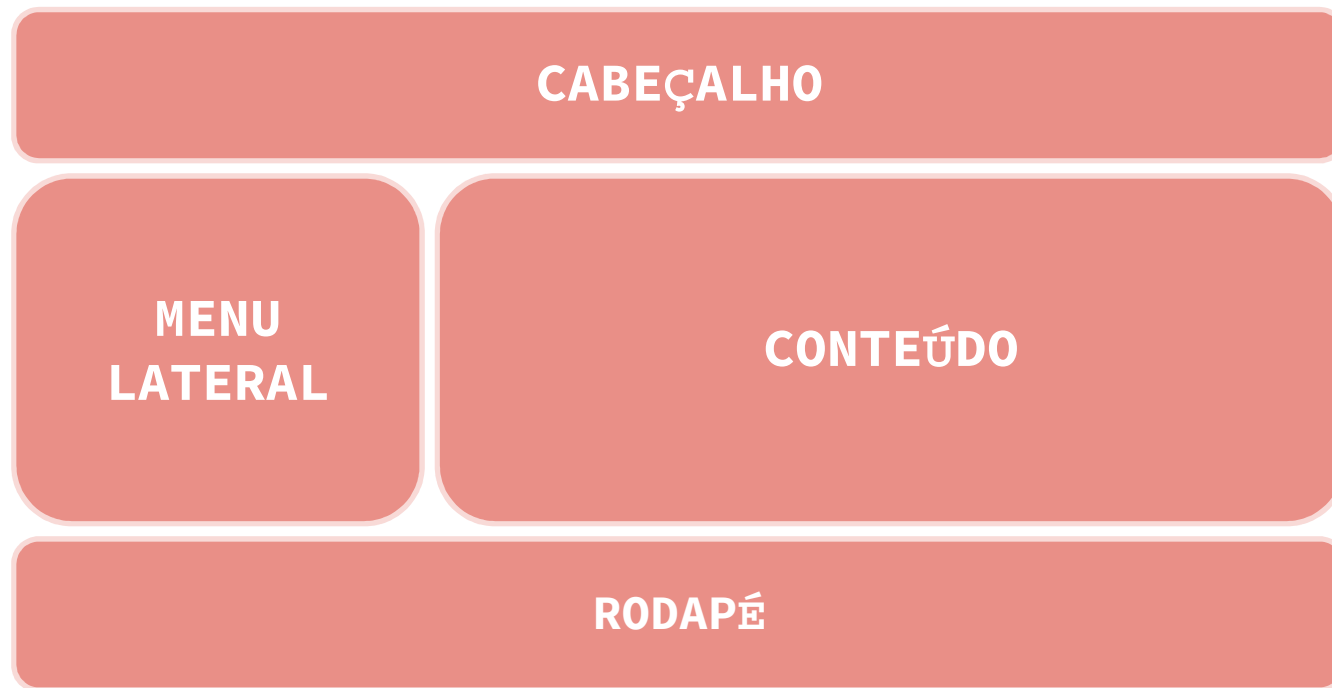
Listener = registra qual método é o “ouvinte” que será chamado quando um item do componente selectOneMenu for selecionado



Modifique a classe `AlunoBean` de modo que ela possibilite utilizar combos aninhados para a UF e o Município do aluno

TEMPLATES

TEMPLATES PARA ORGANIZAR PÁGINAS



TEMPLATES PARA ORGANIZAR PÁGINAS

Passo 1 – Criar a página de layout que define a estrutura que as demais páginas devem seguir

Passo 2 – Configurar a página que será adequada usando o layout definido

PASSO 1 - LAYOUT

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
  <h:head>
    <title>
      <ui:insert name="titulo">
        Sistema Acadêmico
      </ui:insert>
    </title>
    <h:outputStylesheet library="css" name="estilo.css" />
  </h:head>
```

... continua ->

Biblioteca com as
TAGs para criação
de templates

Define uma área do
template que pode
ser alterada pelas
páginas que usam o
template

PASSO 1 - LAYOUT

... continuação

```
<h:body>
  <header>
    <h:graphicImage library="imagens" name="logo2.png" />
  </header>
  <div class="tudo">
    <div id="menu" class="menu">
      <ui:include src="menu.xhtml" />
    </div>
    <div id="conteudo" class="conteudo">
      <ui:insert name="corpo" />
    </div>
  </div>
  <footer>
    Sistema Acadêmico
  </footer>
</h:body>
</html>
```

Inclui dentro da div
a página indicada

Substituirá aqui a
parte chamada de
corpo

PASSO 2 — ADEQUANDO AO LAYOUT

```
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
    <ui:composition template="layout.xhtml">

        <ui:define name="menu">
            <h:commandLink value="Menu" action="menu.xhtml"/>
        </ui:define>
        <ui:define name="titulo">Listar Alunos</ui:define>

        <ui:define name="corpo">
            <h1>Alunos Cadastrados</h1>
            <h:form id="frm">
                <h:dataTable value="#{alunoBean.alunos}"
                    border="1" cellpadding="0">
```

Determina o arquivo
que possui o
template

"define" é usada para
alterar qualquer área
definida pela TAG
insert no template




Faça as modificações nas páginas de modo que todas elas usem o template definido

JSF x JDBC

CRIANDO O LOGIN

O Login é uma página JSF que permite a entrada no sistema

Esta página chama o método `usuarioBean.validar()`, que valida se o usuário é válido ou inválido



Identificador:

Senha:

VALIDANDO O USUÁRIO

```
public String validar() {  
    if(usuario.validar()){  
        this.usuarioLogado = usuario;  
        usuario = new Usuario();  
        return "menu";  
    }  
    return "invalido";  
}
```

Para que é usado o atributo **usuarioLogado**?

COMO FAZ A VALIDAÇÃO?

Login.xhtml

```
<h:panelGrid columns="2">
```

```
<h:commandButton value="Ok" action="#{usuarioBean.validar()}" />
```

```
<h:commandButton value="Limpar" action="#{usuarioBean.limpar()}" />
```

```
</h:panelGrid>
```

COMO FAZ A VALIDAÇÃO?

UsuarioBean

```
public String validar() {  
    if(usuarioLogado.validar()){  
        return "menu";  
    }  
    return "invalido";  
}
```

COMO FAZ A VALIDAÇÃO?

Usuario

```
public boolean validar() {  
    return UsuarioDAO.validar(this);  
}
```

COMO FAZ A VALIDAÇÃO?

UsuarioDAO

```
public static boolean validar(Usuario user) {  
    try(Connection connection = new ConnectionFactory().getConnection();  
        PreparedStatement stmt =  
            connection.prepareStatement(UsuarioSQLs.VALIDAR.getSql());){  
        stmt.setString(1, user.getIdentificador());  
        stmt.setString(2, user.getSenha());  
        ResultSet rs = stmt.executeQuery();  
        rs.last();  
        if(rs.getRow()>=1) return true;  
    }catch(SQLException e){  
        System.out.println("exceção com recursos - validar");  
    }  
    return false;  
}
```




Modifique as classes para cadastrar um aluno no banco de dados usando JDBC